Universitat Autònoma de Barcelona

Facing the Challenge of Automated Negotiation with Humans

A dissertation submitted by
Angela Fabregues Vinent
at Universitat Autònoma de Barcelona
to fulfill the degree of **PhD** in Computer Science.

Bellaterra, September 18th, 2012

Director:	**Prof. Carles Sierra**
Tutor:	**Dr. Josep Puyol**
Elaborated at:	Institut d'Investigació en Intel ligència Artificial Consejo Superior de Investigaciones Científicas (IIIA-CSIC)

Acknowledgements

Voldria agrair a molta gent el temps que ha compartit amb mi aquests darrers anys corresponents a la realització d aquesta tesi doctoral. Especialment, voldria donar les gràcies a en Juan Carlos. Sempre m has acompanyat. Sempre m has ajudat. Sempre has estat allà quan t he necessitat, ja fossis al meu costat o a milles enfora. Ara que nos veim cada dia, esper no te cansis de jo. T estim!

Molts coneixements previs he hagut de menester. Molts altres els he adquirit pel camí. L escola, l institut, la carrera i el màster a l IIIA m han aportat molts d ells. També el meu pas per l empresa privada, per tot arreu s aprèn. Els valors, en canvi, s aprenen a casa. Els vaig aprendre a Menorca gràcies als meus pares i també als meus germans. Papà! Mamà! Me vau donar una infància impressionant, envoltada d un entorn ple de coses per experimentar sentint-me segura i protegida. Me vau deixar créixer, que formés sa meva pròpia personalitat, que anés agafant responsabilitats, i que fos lliure de decidir per jo mateixa lo que ningú més podia decidir. Vau confiar en jo i me vau finançar els estudis a Barcelona. Casi res! Moltes gràcies per tot. Bep, amb tu vaig aprendre a enraonar, a donar mil voltes a ses coses i a poder veure-les des de diferents punts de vista. Molt útil. Joan, company de mil peripècies i experiments. Mira si nos duim bé que anys després hem tornat a viure junts. A veure quina serà sa propera que nos passa. Esper sigui bona.

 Culpables directes de la meva trajectòria acadèmica són en Joaquim Font, en Cerquides, na Maite, na Noyda, en Francesc, n Arcos, i com no, en Carles. Tots heu influit directament en decisions clau de la meva vida acadèmica i professional. Com crec que m han portat per un bon camí, no vull deixar passar l ocasió d agrair-vos-ho. Carles, gràcies per deixar-me gaudir d aquests anys de doctorat. Sé que en alguns moments puc arribar a ser un corc, de vegades és imprescindible. Per les altres em disculpo i t agraeixo la paciència. Diuen que amb el temps els estudiants s assemblen als seus directors. Tan de bo se m hagin enganxat algunes coses de tu.

Companys d aventures, bromes, viatges, moments difícils i celebracions n hi ha molts. Vull esmentar principalment a en Dani i na Txell. Però també a la resta de l IIIA: n Isaac, n Amanda, na Patricia, en Nin, en Jordi, n Anna, en Tito, en Ramón (duré qualque bòtil), en Pedro (traeré queso), en Lluís, na Pilar, en Pablo, na Sandra, en JAR, en Felix, en Marco, n Ana, na Montse, n Imma, na Carol, na Nuria, ... tots no hi cabeu. Segur que m oblido de gent important. Perdoneu. Una de les parts més interessants del doctorant és la realització d estades en altres centres de recerca. Voldria agrair l oportunitat de fer aquestes estades a Londres, Jerusalem i Tel Aviv a: Prof. Mike Luck, Prof. Mark D Inverno, Prof. Jeff Rosenschein i Prof. Sarit Kraus. També vull agrair als seus estudiants i colaboradors, i a amics que he fet per aquests indrets: Samhar, Sanjay, Matthew, Padmaja, Yato i Michelle, Yael, Reshef, Noam, Roy, Roi, Kobi and Raz. Entranyables són també els nostres amics de València: Maria, Elena, Jose, Natalia, Estela, Eva, Carlos i Vicent. Fem tanta pinya que es pensen els de fora que som del mateix laboratori. Bon senyal.

Research supported by the Agreement Technologies CONSOLIDER project under contract CSD2007-0022 and INGENIO 2010, and by the Agreement Technologies COST Action, IC0801.

Abstract

The research field of negotiation has been studied from many different perspectives, among them: game theory, psychology, business, neuroeconomics, and psychopharmacology. The computational study of negotiations is denoted by automated negotiation. Most works on automated negotiation assume rational agents and static negotiation problems. However, humans are rationally bounded, and their negotiations are usually dynamic. It is often impossible to explore the complete negotiation space due to time limitations and the dynamics of the problem. By the time that an optimal solution is found, the solution is not optimal any more. Currently available testbeds on automated negotiation share the same shortcomings. Those testbeds that intend to involve humans in experiments assume that humans are rational, or are defined over artificial domains that require intense instruction of experiment participants. This thesis contributes to automated negotiation defining an agent architecture suitable to negotiate with humans, and a testbed that allows for an easy participation of humans in experiments.

We denote the agent architecture by HANA. It allows multiple bilateral negotiations about actions, and deals with pre-negotiation looking for good enough sets of actions and offers. It is a modular architecture based on an ecological model of rationality. The mental state of the agent is represented as graded beliefs, dynamic desires and general intentions. We use a novel search&negotiation technique where search and negotiation go hand in hand: the former providing offers to propose, and the later providing commitments for pruning the search space, and information for fine-tuning the evaluation of offers. Several negotiation strategies are provided that can be dynamically combined. The architecture is extensible, allowing the incorporation of new behavioural models.

The name of the testbed is DipGame. It is based on a popular board game where being a skilled negotiator is crucial for winning. DipGame allows the study of relationships, emotions, and coalitions that take place during successive negotiations involving humans. There are many research opportunities in different topics all of them connected to negotiation. The study of a topic or another is selected constraining the negotiation language used during the game. The testbed provides a framework for agent development, and several negotiation utilities for the representation of messages and communication among agents. It assists the execution of experiments using a graphical software application called GameManager. It facilitates the inclusion of humans with another application called ChatApp. Moreover, the analysis of results is supported by a different application called DipTools.

This thesis is completed with a formal definition of the problem, a formal specification of the game, and the application of the work to the game industry.

Resum

El camp de recerca en negociació ha estat estudiat des de diferents perspectives. Entre elles: la teoria de jocs, la psicologia, els negocis, la neuro-economia, i la psico-farmacologia. L estudi computacional de la negociació s anomena negociació automàtica. La majoria de les feines sobre negociació automàtica assumeixen que els agents són racionals, i els problemes estàtics. En canvi, els éssers humans són racionalment limitats, i els problemes acostumen a ser dinàmics. Sovint resulta impossible explorar l espai de negociació complet degut a l esgotament del temps i al dinamisme del problema. En el moment en què es troba una solució òptima, aquesta ja ha deixat de ser òptima des de fa temps. Els actuals bancs de proves disponibles sobre negociació automàtica es troben amb els mateixos problemes. Els que pretenen ser compatibles amb agents humans assumeixen que aquests són racionals, o utilitzen dominis artificials que requereixen una instrucció intensiva dels éssers humans per tal que puguin participar en els experiments. Aquesta tesi contribueix a la negociació automàtica definint una arquitectura d agent adequada per a negociar amb els humans, i un banc de proves que resol els problemes existents a l hora d incloure humans en els experiments.

L arquitectura d agent s anomena HANA, permet múltiples negociacions bilaterals sobre accions a realitzar, i s ocupa de la pre-negociació tot cercant bons conjunts d accions i ofertes. Es tracta d una arquitectura modular basada en un model ecològic de la racionalitat. L estat mental de l agent es representa amb graus de creences, desitjos dinàmics i intencions generals. Utilitzem una nova tècnica de cerca&negociació on la cerca i la negociació van de la mà: una proporcionant ofertes per a proposar, i l altra compromisos per a podar l espai de cerca, i informació per a afinar l avaluació de les ofertes. Es defineixen diverses estratègies de negociació que es poden combinar dinàmicament. L arquitectura és extensible permetent la incorporació de nous models de comportament.

El banc de proves s anomena DipGame i es basa en un joc de taula molt popular on ser un bon negociador és crucial per a guanyar. Aquest banc de proves permet l estudi de les relacions, les emocions i les coalicions que tenen lloc durant successives negociacions entre éssers humans. Hi ha moltes oportunitats d estudi en diversos temes de recerca, tots ells vinculats a la negociació. L estudi d un tema o d un altre es selecciona restringir el llenguatge utilitzat durant el joc. El banc de proves proporciona un marc pel desenvolupament d agents i diverses eines de negociació per a la representació dels missatges i la comunicació entre ells. DipGame dóna suport a l execució d experiments utilitzant un programa anomenat GameManager, i facilita la inclusió dels éssers humans amb un altre programa anomenat ChatApp. A més, es dóna suport a l anàlisi dels resultats amb un programa diferent anomenat DipTools.

Aquesta tesi es completa amb una definició formal del problema, una especificació formal del joc i l aplicació del treball a la indústria dels jocs.

Contents

List of Figures

Chapter 1

Introduction

Science is as diverse as the universe it aims to study, to model, to change. Artificial sciences concentrate on the art of design. This thesis belongs to engineering as it focuses on the design, building and the use of a particular kind of machine: a machine with artificial intelligence for negotiating with humans.

There is no standard definition for *artificial intelligence*, neither for *intelligence*. We follow the definition in [Russell and Norvig, 2003], and classify the research done in this area by its goal in two dimensions: (1) whether it concerns *thinking* or *behaving*; and (2) whether it is done *rationally* or *as humans do*. Basically, works on robotics concentrate on the behaviour, and works with no physical components focus on thinking. The distinction between being *rational* or as *humans* imply that humans are not assumed to be completely rational. In fact, they are *bounded rational* as it is not often the case of having all the information and the necessary time for making rational decisions [Simon, 1996]. As stated in [Smith, 2003; Debenham and Sierra, 2009], humans do not follow a *constructivist* sort of rationality. For instance, human decisions depend a lot on e.g. the relationships with the other humans [Sierra and Debenham, 2007], on emotions [Fessler and Haley, 2003], and on cultural influences [Gal et al., 2010]. The correct classification of this thesis is on *thinking as humans*. Thinking as humans is necessary to understand them and, thus, succeed negotiating with them.

We use the term *agent* to refer to an autonomous entity that observes and acts upon an environment directing its activity towards achieving particular goals. Some of those agents are denoted by *intelligent agents* and can be humans or software agents. In this thesis, we are specially interested in groups of those agents interacting in the same environment, that is, we are interested in MultiAgent Systems (MAS) research. The machines that we design are indeed intelligent software agents with no physical components.

1.1 Motivation

Multiagent systems research studies problems involving multiple agents and their environment. For instance: resource allocation, timetabling and team planning. The most common industrial solutions to those problems are centralised, like [Alvarez-Valdes et al., 2002; Bardadym, 1996]. The application of these solutions in domains that exclude human interaction is already a reality. That is also the case in those domains where a human is the user of a decision support system controlling all the process. The next step is to look for solutions to problems that include several humans interacting. In those problems, we true believe that negotiation can improve the outcome of agents allowing a better usage of resources. And, mainly, it will let humans contribute providing missing knowledge about their preferences, and it will let them feel more likely to accept the final solution. Example 1 describes a problem of scheduling optimisation where several humans try to adjust the scheduling to their own preferences.

Example 1 *Juan Carlos is the head master deputy*[1] *of a high school. He has a chemistry degree, and he is not instructed on timetabling. However, he has to set the timetables for all the teachers and students of his high school. Fortunately, there are software applications that can deal with what seemed to be the big problem. Now, the big problem is to deal with teacher's change requests on the timetable that is arbitrarily generated by the computer for them.*

Being intransigent to change requests is the fastest and easiest solution. However, faster and easier does not necessarily mean better. An imposed strict solution can imply many unnecessary troubles in personal life. Big companies often use as a slogan that they facilitate *the reconciliation of family and working life.* Happy parents seem to mean happy workers. Being it one of the principles of Juan Carlos high school, he should facilitate parents to take their children to school by accepting, as long as it is possible, requests from teachers with young children about having their first time slot of the day empty. As this issue with the first time slot, many other issues may arise. Letting humans to take part in the process helps them also to take conscience, and facilitates the acceptance of the final solution. Negotiating to exchange time slots with colleagues is a good way to refine the solution provided by current industrial products.

What we propose is to study what we denote by *Resource Negotiation Problem* (RNP), that is a general multiagent problem where negotiation can be used to reach a distributed solution, or to refine a poor centralised solution, without reviling the agents preferences. We define the problem with the aim to be as general as possible and facilitating, at the same time, to run experiments on it with negotiations involving humans. For that reason, we took a look to the most common human day life negotiation scenarios and avoided some features that complicated too much the experiments. For instance, in RNP the physical actions are fully observable whereas, in real life, humans can often observe only some of them. Another simplification is to constrain the execution of actions to be executed only in concrete time instants.

Software agents in this setting can supplant humans using knowledge about the human individual and its preferences to negotiate with the other agents. Someone could prefer to delegate on a trusted software agent that will negotiate on its behalf without loosing privateness of information. In this timetabling example, imagine a teacher named Beth that does not want to teach the naughty son of her friend. She would like to avoid having him as pupil although she does not want to tell anybody why. A centralised solution of the problem cannot satisfy Beth s need. Letting to refine timetabling solutions representing them as RNPs would let her negotiate time slots keeping her privacy intact. The idea is

[1]In a high school, the head master deputy is a teacher that coordinates the rest of teachers.

that Beth could either negotiate by herself or use a software agent on her behalf knowing all this information, and her willingness of never sharing it. She does not need to tell Juan Carlos that she hates that boy or that she is worried about having the son of a friend as pupil.

The field of negotiation is very broad. Why humans cooperate and how they reach agreements has been since long a fascinating area of inquiry. It has been studied from many different perspectives: game theory [Raiffa, 2002; Kraus, 2001], psychology [Adams, 1965; Sondak et al., 1995], business [Bazerman et al., 1992], neuroeconomics [Coricelli and Nagel, 2009], or psychopharmacology [Eisenegger et al., 2010] just to mention a few. The computational study of negotiation is denoted by *automated negotiation* and belongs to the MAS research area. A very complete analysis of the challenges of negotiating autonomous computational agents is included in [Lopes et al., 2008].

An approach to automated negotiation is to study communication protocols with properties that promote a particular behaviour among the agents involved in the negotiation [Rosenschein and Zlotkin, 1998]. These protocols are highly restrictive and assume a rational behaviour of the agents that cannot be assumed when humans can take part of the negotiation. Another approach focuses on the negotiation strategies of the agents, [Sycara, 1989; Kraus, 1995b; Faratin et al., 1998a; Sathi and Fox, 1989]. This allows the use of more flexible protocols, and the inclusion of humans as negotiation counterparts because, in this case, agents are not assumed to be fully rational. The first approach guarantees optimality exploring all possible deals, whereas the second approach takes into account the cost of the deal exploration, and uses heuristics to look for good-enough deals. A heuristic approach takes into account the complexity of algorithms, and aims to provide a feasible solution instead of looking for a probably very costly optimal solution. Perfect is the enemy of the good , Voltaire. On one hand, this approach is based on realistic assumptions facilitating its application to real problems and providing more flexibility for agent design. On the other hand, it needs extensive evaluation through empirical analysis. In both approaches, the negotiation consists of trading proposals. No other information is exchanged between agents. No opinions, explanations or arguments pro and against particular proposals. It is broadly accepted that argumentation facilitates the signature of agreements. The problem is that the communication protocols needed for argumentation are complex. Argumentation based works are theoretical, and difficult to apply [Pardo et al., 2011; Jennings et al., 2001].

We are particularly interested in the heuristic approach due to its suitability for human agents, and its possible practical applications. However, facing the challenge of automated negotiation with humans means a lot more than defining a nice negotiation strategy. It means to build an agent that quickly reacts to the observations of the environment, that can measure its utility and the utility of the other agents, that can search for good enough proposals, understand the relationships among agents, their reactions, ... Pre-negotiation is the term used in works that deal with all what is necessary for negotiation. As stated in [Munroe, 2005], many works on automated negotiation assume negotiation to be an isolated endeavour not affected by the broader scope of an agent s activity and scope. Contrarily, negotiation environments in real-world applications involving humans do not use to be isolated. Humans use previous interactions to decide whom to negotiate with and what offers to propose. For instance, if someone never honoured an agreement in the past, we will not even start a negotiation with him/her as we will not expect him/her to honour any new agreement in the future. Trust is essential for negotiation to take place [Johnson and Luecke, 2005].

The majority of work on automated negotiation assume, for simplicity, that during a whole negotiation session:

1. The negotiation domain establishes a fixed set of issues and ranges of values per issue,
2. Negotiation outcomes and negotiation objects are mappings of values assigned to issues,
3. Agents are aware of their preferences on negotiation outcomes,
4. Preferences can be represented as a utility function that assigns a numerical value to each negotiation outcome,
5. Preferences remain static.

It has been proved that humans have difficulties to represent their preferences [Domshlak et al., 2011]. It is quite difficult to imagine them having a utility function assigning a numerical value to each negotiation outcome. However, more difficult to achieve for them is keeping those preferences static during the whole negotiation session. Humans may change their criteria a bit during the course of the negotiation. This is specially the case if we understand negotiation as the process towards the signature of agreements. If we consider that negotiation is a crucial agreement technology [Luck and McBurney, 2008], we cannot intend agents to remain indifferent to the external changes in the environment and to the changes given by the information provided by the negotiation. Assuming fixed preferences would mean to assume that: every agent has the complete true reality, and negotiation is a simple matter of coordination instead of a process to reach agreements.

We think that negotiations must be seen as part of an agent s (execution) life and not as the unique and ultimate goal of it. In fact, several negotiations may take place during the life time of an agent while the internal state of the agent would change according to the current situation and the previous experiences. Thus, we follow Sierra and Debenham s work [Sierra and Debenham, 2007] and take the context, the relationships with the other agents, the previous interactions, and commitments all into account.

The design of an agent requires a lot of work in many different aspects of the agent. Several research topics are involved in such task starting with *agent architecture design* and moving from *search* to *negotiation* maybe through *trust* or *emotions*. The agent can be as complex as desired including results from many different areas. We think that a good way to proceed is to start with a basic agent and to keep upgrading it, adding more capabilities to it. Thus, the construction of an agent is done by the combination of capabilities. We facilitate the design of a negotiating agent involved in an RNP providing the HANA agent architecture specially designed to allocate all those capabilities desired to negotiate with humans.

The evaluation of negotiation strategies is empirical and require the execution of intensive experiments. There is a chronic lack of shared application domains to test advanced research models and agent negotiation architectures in multiagent systems [Fabregues and Sierra, 2009]. Specially when the negotiation domain and the agent preferences are not fixed during the negotiation session as it is assumed by [Lin et al., 2011]. Moreover, there are recognised difficulties of running experiments on negotiation involving both human agents and software agents [Bosse and Jonker, 2005]. These difficulties are delaying the production of automated negotiating agents. First, most research work on negotiation techniques does not consider humans as counterparts of automated negotiating agents. Second, enticing humans to participate in negotiation experiments is difficult because the negotiation environment use to be artificial and not attractive, and because the language to use in interactions is unnaturally constrained. To test the performance of agents, following, for instance, the HANA architecture, we need a common domain being a specific instance of RNP. The domain must be deterministic, avoiding random elements that would add noise to the results. It must permit several levels of difficulty, and it must be able to be applied to the industry. But mainly, it has to be attractive for humans to take part in the experiments as, otherwise, congregating them would be too difficult. Gamification seems to be the solution [Deterding et al., 2011]. A well known deterministic game satisfying the previously listed properties would be a good domain. A game does also imply a direct application to the game industry providing software agents as software players. Moreover, humans that are fans of the game would be glad to take part in the experiments. We propose to use The Diplomacy Game as the domain for experimentation.

In order to run experiments using a particular domain, it is necessary to specify the domain formally, and analyse its complexity. The disposability of strategic and tactic studies of the domain would be appreciated for the creation of heuristics. The Diplomacy Game is quite popular, and there are several works studying strategies and tactics specific for this domain. However, there was no previous formal specification of the game. In this thesis, we formalise the game and provide a complexity analysis. We even go further and provide a complete infrastructure with software for managing the game, collecting results, supporting the analysis of those results, and we also provide a framework for building software agents for playing the game. All those resources form the DipGame testbed.

One of the purposes of this thesis is to empower technology transfer. Most current work on MAS, and specially on automated negotiation, are rather theoretical, concerning artificial problems, and making strong assumptions that make difficult the application of their results to the industry. Even the most practical work has no real applicability. Despite the fact that Diplomacy is indeed a game, it is a realistic problem due to its complexity, the importance of negotiation and relationships in it, and because humans are also part of the problem. The Diplomacy Game can be seen as a specific instance of an RNP. Being a popular game does simply aggregate benefit as it is a well studied problem, with many people interested in it, and with many people that can potentially take part of the experiments. To congregate all those people and, at the same time, to apply our work to the game industry, we developed a web application for playing the game online against software players. Moreover, we have collaborated with a mobile game company integrating our agents in their system and, by this, allowing their users to play against our software agents from mobile devices.

1.2 Contributions

This thesis contributes to the research area of MAS and concretely to automated negotiation:

First - Designing an agent architecture Recently, an increasing interest on the psychological and cultural aspects of negotiation has been observed in the automated negotiation field. For instance, agent negotiation architectures inspired in human relationship building [Sierra and Debenham, 2007], or analysis of cultural influences in negotiation strategies [Gal et al., 2010]. The community of social simulation has also studied negotiation models in as much as they help in defining policies (e.g. [Hales, 2002] in an environmental setting) or norms [Conte and Sichman, 1995]. These human-oriented approaches follow an *ecological* model of rationality [Smith, 2003] based on two basic notions: *everything* in the world is constantly changing and not *all* facts can be known by an agent. This view on rationality is in line with an evolutionary approach: knowledge evolves, individuals evolve and societies evolve. Ecological rationality, observed in humans and human societies, seems also the right one when we face the design of *software agents* that aim to negotiate with humans. If knowledge is dynamic and evolves along time, this will certainly be also the case for beliefs or intentions of agents that will change constantly due to the interaction with the environment and the other agents.

We have designed HANA as a modular agent architecture suitable for RNP with a huge space of possible solutions and incomplete information. It is capable of multiple bilateral negotiation, and takes pre-negotiation into account modelling other agents, and generating negotiation options on the fly. The architecture contains a world model with graded beliefs and desires that trigger graded intentions. It is based on g-BDI where degrees are used to represent the uncertainty on the world, [Casali et al., 2011]. In this architecture, desires are not fixed, they change guided by the agent s emotions that respond to the observation of the world represented as beliefs. The architecture is extensible allowing behavioral models to be incorporated in the BDI providing, for instance, trust evaluations or particular rules to generate intentions. Intentions are general, and trigger the decision making of the agent that is included in the negotiation module. What actions to perform and messages to send is decided by the negotiation strategy taking into account the best found action plans and the best found acceptable proposals.

The architecture relies on a bounded rationality, and it assumes that the space of solutions is large enough, and the time for deciding is short enough, that looking for optimality is practically unachievable. To cope with this lack of time for searching we provide a new technique for search&negotiation that is reflected in the plan search module. The idea is to use an anytime algorithm providing a ranking of good plans, and to use negotiation to fine tune the evaluation of those plans obtaining information from the negotiation of proposals of actions included in those plans. Thus, search and negotiation go hand in hand: the first providing interesting proposals from plans, and the second providing information to improve the evaluation of those plans. We also provide a set of negotiation strategies that can be combined in order to take advantage of the whole diversity of information included in the world model. Notice that, as behavioural models can be incorporated to the world module generating intentions, and those intentions trigger the decision making of the agent, that is located in the negotiation strategy; then the negotiation strategy can be enriched with a huge diversity of information instead of being forced to work with a black box utility measure. The architecture is suitable for dynamic environments that change during the negotiation process. The constant update of the world module given new observations, and the concurrent execution of the plan and proposal search, allows the negotiation strategy to dispose of preprocessed up to date information.

Second - Developing a testbed Negotiation models that have been designed in the past were validated with simple toy environments artificially created by the same researchers that design the models. The community requires a more realistic scenario where negotiation models can be tested. The problem with artificial scenarios is that the behaviour of humans is not necessarily the same as in real life. It is not the same to decide on artificial money or on real money. That is why areas like auctions have moved into more realistic scenarios like the different TAC competitions, or why areas like neuroeconomics [Sanfey et al., 2003] or experimental economics in general have moved into real scenarios. Current testbeds [Hindriks et al., 2009; Gal et al., 2005; Wellman and Wurman, 1999] are rather simple and we believe that the validation of negotiation models that aim at human-agent interactions [Amgoud and Prade, 2009; Alsinet et al., 2008; Sierra and Debenham, 2007] require more sophisticated testbeds.

We developed the DipGame testbed using The Diplomacy Game as the domain for experimentation. The scenarios produced by this game include the interaction with several human agents that compete, but that can occasionally cooperate. The negotiation is crucial in the game that checks the negotiation capabilities of the players. This testbed allows the study of relationships, emotions, and coalitions that take place during successive negotiations. There are many opportunities in different research topics all of them connected to negotiation. The study of a topic or another is selected constraining the negotiation language and protocol.

We have defined L, a language hierarchy with increasing levels of complexity, to illustrate those opportunities, and establish a standard language for DipGame. The hierarchy allows researchers to gradually progress in the task of designing a negotiating agent. Levels correspond to increasing complexity. Argumentation is included in L as the highest language level. The domain dependent vocabulary is separated and represented as an ontology. One of the aims of this testbed is to facilitate the posterior application of the work to the industry. The separation of the Diplomacy vocabulary simplifies the reuse of code to that end.

The testbed infrastructure includes several negotiation utilities for the representation of L messages and the communication among agents. It even includes a software library that translates messages between L and (a restricted set of) English. It analyses the dialogue to complete sentences. This library is called dialogueAssistant, and it is very useful for human agents that, by this way, do not need to be instructed in order to take part of the experiments.

For the development of agents to run on DipGame, the testbed provides a framework that copes with the domain dependent issues like the map and game state representation, the control of the dynamics of the game (years, phases, orders required, end), the representation of moves (orders), and the negotiation dynamics (received messages, messages allowed to be send). A methodology for developing agents and a tutorial are included in this thesis.

DipGame assists the execution of experiments using a graphical software application called GameManager. This application assists the setting and execution of experiments. It allows the loading of new software agents, the execution of games with agents running in several machines, the gathering of execution logs online, and its saving in a file.

For humans to take part in the experiment, a graphical software application, called ChatApp, has been developed. The application can run as an standalone application or embedded into the GameManager. It represents the game state over an interactive map that human users can use to indicate their movements. A chat is included to allow the negotiation with the other players. The chat uses the dialogueAssistant library to represent messages in English and translate them to L. Several usability and aesthetic techniques have been used to please the users.

The analysis of results is supported by a different software application called DipTools. This application reads the data from game logs stored in files by the GameManager. It provides intuitive graphical tools for the representation of these data. The main tool is a chart representation of several variables (selected by the user). They can be represented during the process of a game, comparatively between several games, and comparatively between several groups of games. Aggregative operations, like the mean, can be used to simplify those charts. DipTools facilitates the analysis work to be done by researchers after running an experiment. A methodology for running experiments with DipGame is also included in this thesis.

As part of the main contributions introduced above or completing them, there are other contributions that we list below:

Third - Problem formalisation The formalisation of the problem is necessary for the definition of the HANA architecture as it specifies the environment and the negotiation protocol that is assumed by the architecture. The problem, denoted by Resource Negotiation Problem, is a resource allocation problem that specifies a negotiation protocol for agents to negotiate about the performance of actions.

An environment is defined as a partition of resources among agents. Agents controlling resources operate on them performing actions that drive the evolution of the environment. The negotiation is limited to sets of actions that can be proposed, accepted, and rejected. The protocol allows for multiple bilateral negotiations. An agent cannot send a proposal to someone that is waiting for its reply. But it can send a proposal to someone else. The idea is that proposals must be replied. However, several negotiation dialogues can take place simultaneously involving a given agent. Actions take place in specific time instants. In those instants, all agents perform their actions at the same time. The negotiation proceeds between each pair of successive instants. All actions being performed are observable, whereas negotiations remain private. We define RNP and provide a simple example.

Fourth - Specification of The Diplomacy Game The Diplomacy Game is a quite popular board game.[2] There are books about the strategy and tactics of playing Diplomacy [Sharp, 1978b,a], an online magazine [zin], and many clubs and tournaments [tou]. The rules of the game are described in the rule-books included in the commercial versions of the game. In addition, there is a guide for Diplomacy experts that describes an algorithm for resolving the moves [Black et al., 1998]. The article [Kruijswijk, 2009] discusses the rules in terms of mathematics, but still there was no formal specification of the game previous to this thesis. The use of this game in a research context requires the existence of a formal specification that gives a precise description of the system that is going to be used. We specify The Diplomacy Game using the Z-Notation formalism. The specification describes the rules of the game that include its dynamics, and the algorithm for resolving moves and generating the next turn. In addition, we analyse the complexity of the game and represent it as an RNP.

Fifth - Application to the game industry Several online game solutions appeared in the last years for playing Diplomacy: *webDiplomacy*, *playDiplomacy*. We decided to provide a new one where software agents could take part of the game. We created the DipGame web application by which we join research and entertainment together as it was previously successfully done in other projects.[3] Thus, we provide an online game industry product that allows Diplomacy players to take part of games facilitating the congregation of players, and providing the option of including software players (bots) to complete the number of players necessary for a game. Therefore, players are happy to play, and researchers are happy to test their software agents playing against humans. The host of this web application is the DipGame website that we have created to disseminate the testbed, and provide its documentation and software resources online.

The recent implantation of smart phones and the increasing use of mobile devices for entertainment have motivated the need to play Diplomacy using a mobile device. The development time required for creating such application stopped us for a while from doing it. Fortunately, Droidippy appeared, and the mobile game company powering it agreed to collaborate with us. Droidippy is a product that allows both to play online using a web browser, and to play from a mobile device using an Android application.[4] The collaboration consists in integrating our work in their system letting DipGame agents to take part of Diplomacy games running in Droidippy.

1.3 Overview

This thesis is structured in several chapters. In general, we assign a chapter per contribution, and introduce them in this thesis in a natural order. We start with the background introducing concepts and describing related work, and

[2]Diplomacy has a board game rank of 250 ranked by 6486 players at Board Bame Geek: `http://www.boardgamegeek.com/boardgame/483/diplomacy`.

[3]In 2005, Luis von Ahn devised the ESP Game, an online game of image labelling that Google is now using to improve its image search results.

[4]Web browsers for mobile devices enable the possibility to play using the web interface on mobile devices. Those devices use to have small screens. The use of mobile dedicated applications is recommended in order to use as much screen area as possible.

end with a discussion of the contributions of this thesis comparing them to the related work. A summary is included at the end of each chapter. The structure in chapters of this thesis is as follows:

Chapter 2: Background This chapter describes the previous work strongly related to this thesis. We do not provide a survey on automated negotiation. We consider that the whole research field is successfully described in [Lopes et al., 2008]. We focus on multiple bilateral negotiation and pre-negotiation as those are the most relevant aspects of our architecture. The testbeds related to automated negotiation are described as well as the background on The Diplomacy Game, and the agents that succeeded negotiating to some extend while playing Diplomacy. We conclude describing the Z-Notation formalism used to specify the game.

Chapter 3: Resource Negotiation Problem In this chapter, we formalise the general problem that is the purpose of study in this thesis. An RNP is a particular resource allocation problem where several agents control the resources of the environment and perform actions on it. Those actions guide the evolution of the environment. The environment is defined as well as the notion of plan, state and the negotiation protocol. Then, the problem is illustrated in an example.

Chapter 4: The Diplomacy Game The Diplomacy Game is a popular strategy board game played by seven players that perform moves concurrently, it is not a turn taking game. As a consequence of the concurrency of moves, conflicts emerge. The rules of the game establish a method for resolving those conflicts, and proceeding generating the new game state for the next turn. This algorithm is denoted by the adjudicator algorithm. We describe the game, its players community and the existing software agents. We also explain that turns are identified by a year and a phase corresponding to a particular season.

In this chapter, we provide a formal specification of the game in Z-notation. We start specifying the static part of the game. Then, we proceed with the dynamic part, defining the game state and how to initialise the game. Next, the adjudicator algorithm is specified in parts corresponding to each season. Finally, we analyse the complexity of The Diplomacy Game, and provide a representation of the game as an RNP.

Chapter 5: HANA Architecture HANA is a software architecture for agents that need to negotiate joint plans of action in realistic scenarios such those in RNPs. These negotiations may involve humans and repeat along time. The chapter describes the agent architecture, and its main modules that are: the world model, the plan search, and the negotiation.

Chapter 6: DipGame testbed The DipGame testbed is the infrastructure for software agents to be created and experiments to be executed using The Diplomacy Game. The chapter starts motivating the use of Diplomacy for research purposes. Then, we describe the infrastructure, and define the language. The infrastructure consists of several components to assist the creation of bots, and others to assist the execution of experiments and their analysis. An experiment proving the suitability of the testbed to test negotiating agents is also included.

Chapter 7: Application to the game industry The selection of a game as the domain for experimentation offers the possibility to apply our work to the game industry building software agents capable to play. We have applied our work to the online game industry providing a solution for playing Diplomacy online. Moreover, we have integrated our agents into Droidippy, a mobile solution for playing Diplomacy. To disseminate the DipGame testbed, we have created a website with all documentation and resources of the testbed. This website is also the host of our solution for playing Diplomacy.

In this chapter, we start motivating the application of the work to The Diplomacy Game, and proceed describing the DipGame website and their sections: the player section with the online game industry solution; the researcher section with the testbed documentation and resources; and an analysis of the impact of the website since its publication. Then, we describe the Droidippy integration of DipGame.

Chapter 8: Discussion In this chapter, we discuss the thesis analysing the contributions, and comparing them with the background. Then, we mention some future work research, and conclude listing our publications that are related to the work described in this thesis.

Annex A: Quick start guide This chapter contains a tutorial for the creation of software agents using DipGame.

Annex B: Notation summary This chapter contains a table summarising symbols included in the definition of the RNP and HANA.

Chapter 2

Background

This chapter describes the previous work strongly related to this thesis. We do not provide a survey on automated negotiation. We consider that the whole research field is successfully described in [Lopes et al., 2008]. We focus on multiple bilateral negotiation (Section 2.1) and pre-negotiation (Section 2.2), as those are the most relevant aspects of our architecture. The testbeds related to automated negotiation are described in Section 2.3 as well as the background on The Diplomacy Game (Section 2.4), and the agents that succeeded negotiating to some extend while playing Diplomacy (Section 2.5). We conclude, in Section 2.6, describing the Z-Notation formalism used to specify the game.

2.1 Multiple bilateral negotiation

The work on automated negotiation can be classified by the type of negotiation protocol being used. Common types are: multilateral protocols, typically organised as auctions among more than two agents; bilateral protocols, often with alternating proposals between two agents; and multiple bilateral protocols that are not as popular as the other two.

Multiple bilateral negotiation protocols allow the negotiation of agreements among several, more than two, agents at the same time. The main difference between multiple bilateral and multilateral protocols is that several agreements can be negotiated in multiple bilateral protocols while only one can be negotiated in multilateral protocols. The typical multilateral setting is an auction where an item is auctioned among all the bidders. In this case, the agreement being negotiated, that is the item and the price, is the same for all bidders. It changes through the pass of time given by the auction dynamics, but it is always the same price for all the bidders. Contrarily, in multiple bilateral negotiations, the agent corresponding to the auctioneer can deploy different strategies when bargaining with different types of bidders. This variability means negotiation can be tailored to the individual opponents rather than derived implicitly through the competition of the bidders, as happens in the traditional auctions, [Nguyen and Jennings, 2005]. We think that progress in this research line opens a new branch of possibilities for industrial applications. Specially those providing a personalised attention to the client.

Multiple bilateral negotiations are common in business [Bichler et al., 2003], however automated negotiation research lacks of these kind of negotiations. The main problem here is that many bilateral negotiations take place in an independent manner, although the reached agreements affect a common agent. Thus, controlling those bilateral negotiations is crucial to avoid inconsistent commitments, and to reach the best possible outcome. Previous work on multiple bilateral negotiation combine two protocols. They define a negotiation process with two phases: a bilateral phase, and a multilateral phase. An example is Moai [Raisch, 2002], that allows a negotiator to switch from an auction protocol to one or more bilateral negotiations, [Neumann et al., 2003]. Another example is [Shakun, 2005], where multiple negotiators conduct bilateral multi-attribute negotiations followed, optionally, by an auction. An alternative is to use sequential bilateral negotiations as an approximation to multiple bilateral negotiations [Pinho et al., 2004]. This approach avoids the simultaneity of negotiations and simplifies the problem. It does not negotiate concurrently, however it takes into account a scenario with several agents, and do not consider to be in an isolated bilateral negotiation. The outcome of negotiating with an agent affect the subsequent negotiations with that agent and others. This is, in fact, one of the claims of pre-negotiation that is described in next section.

There are works that simplify the problem assuming that there is only one agreement as an outcome of a negotiation with multiple opponents, [Van de Walle et al., 2001; Nguyen and Jennings, 2004; Rahwan et al., 2002]. In [Van de Walle et al., 2001], the authors propose a fuzzy set-theoretic approach to the analysis of alternatives in multiple bilateral negotiations for the ranking of negotiation counterparts. The idea is to recommend some potential buyers to negotiate with. They use a fuzzy-relational approach to obtain a partial rank-order of the prospective buyers.

Our interest in HANA is in domains where multiple potential agreements could be feasible. In [Vahidov, 2008], the authors share the same interest and envisage a negotiation support systems solution combining human judgement capabilities with autonomous agents. Agents are in charge of every negotiation instance while the human manages the fleet of the negotiation using a graphical interface. Similarly, other works on automated negotiation use an agent per negotiation counterpart and a control or management mechanism (maybe another agent) to control all those agents. For instance, in [Rahwan et al., 2002], an agent is used to coordinate the negotiating agents that conduct reasoning by using constraint-based techniques. They use two levels of strategies: the individual negotiation level, and the coordination level. After each negotiation cycle, the individual agents report back to the coordinator that evaluates their performance and issues new instructions accordingly. The idea of a MAS performing as a single negotiator is not new. The design of The Diplomat, described in Section 2.5, does already assume the existence of multiple agents. The authors do not only assume one per negotiating counterpart and a controller, they assume the existence of many other agents for several parts of the reasoning of the automated negotiator. Our approach skips this tendency of using several agents to build an automated negotiator, and use a single one with a single mental state and with the capability to perform concurrent tasks like, for instance, searching for good possible agreements and sending proposals.

Our alternative to the use a MAS as an automated agent consists on executing those concurrent tasks using several threads sharing the same information and reasoning mechanisms. This alternative can be seen also in [Nguyen and

Jennings, 2005; Williams et al., 2012]. Those works use a negotiation protocol completed by a normative system where, apart of proposing (*offer*), accepting (*accept*) and rejecting (*end*) agreements, agents can *confirm* and *decommit*. Confirm is used to confirm an acceptance. This is necessary in this work to avoid the acceptance of inconsistent agreements by the negotiating threads. The delay between accepting and confirming an agreement can be used by the negotiation counterpart to end the negotiation. However, when an acceptance is confirmed, any decommitment performed by the agents, meaning to break the agreement, implies a penalty. Our approach uses a far simpler protocol where acceptances do not need to be confirmed, and decommitment is not announced. The penalty of decommitment is not stipulated by the protocol, it is decided and performed by the other agents, for instance, not relying any more on the given agent.

2.2 Pre-Negotiation

Most negotiation models address the issues associated with the design of negotiation protocols and negotiation strategies. However, few of them deal with the preparation and planning for negotiation. According to [Saunders, 1996]:
Peace requires a process of building constructive relationships in a civil society not just negotiating, signing, and ratifying a formal agreement. . As relevant is the negotiation protocol or strategy, as it is the structure of relevant information, the analysis of possible negotiating options, and the plan to follow. This task is denoted by *pre-negotiation* and it is studied in other works like [Munroe, 2005; Nguyen and Jennings, 2005; Lopes et al., 2002; Li et al., 2004; Zeng and Sycara, 1998]. We do pre-negotiation in this thesis with the definition of the HANA architecture that generates negotiating options from joint possible plans.

As described in [Lopes et al., 2008], pre-negotiation addresses the operational and strategic process of preparing and planning for negotiation. This concerns the structuring of personal information, the analysis of the opponents, and the definition of the protocol and selection of the initial strategy. For simplicity, HANA assumes the use of a multiple bilateral protocol. Even though, the codification of the protocol as a normative system incorporated in the world makes it possible the use of other protocols. The main restriction in our architecture is on the negotiating objects that are assumed to be sets of actions refusing the possibility to argue about the suitability of a particular negotiation protocol. However, the rest of pre-negotiation activities are present in our architecture.

Definitively, the negotiation model that inspire our work is LOGIC. In [Sierra and Debenham, 2007] the authors focus on bilateral negotiations and claim that the data necessary in a negotiation process can be structured along five dimensions: (L) **Legitimacy**, the relevant information that might be useful to the other agent in the negotiation; (O) **Options**, the possible proposals that are acceptable; (G) **Goals**, the objectives of the negotiation; (I) **Independence**, what can we do if the negotiation fails; and (C) **Commitments**, the previously signed agreements. The data along the LOGIC dimensions are to be prepared before every single negotiation process starts. The evaluation of those dimensions is done based on the utility and the information provided by a given negotiation dialogue. They defined the *intimacy* of a relationship between two agents as the pair of 2×5 matrices evaluating how much utility and information every agent has provided on any single dimension (a matrix per agent). Based on previous psychological studies, they assume that the fairness of a relationship depends on the balance and on the social relationship between the agents that is represented by the intimacy level. The LOGIC agent acts in respond of needs preparing the LOGIC dimensions data and generating a *negotiation target*. Then, a *relationship strategy* determines who to negotiate with. The relationship strategy generates the *relationship targets* expressed as the desired level of intimacy to be achieved in the long term. Those targets are taken into account for selecting the negotiation counterpart. Next, the *negotiation strategy* determines the current set of Options computing the probability of an option being acceptable based, among others, on the degree of *altruism* and the *strength* of the need. If the option set is empty, the agent quits the negotiation withdrawing. *Negotiation tactics* select options to be proposed. They specify the steps to be followed towards a negotiation and a relationship target.

2.3 Testbeds

A testbed is a platform for experimentation. The existing testbeds for automated negotiation can be classified given the communication protocol that they use. There are two testbeds dealing with multi-lateral negotiations. One of them is the Multi AGent NEgotiation Testbed (MAGNET) that, in spite of its name, is rather a generalised market architecture for multiagent contract negotiation using auctions [Collins et al., 1998]. The other is the Trading Agent Competition (TAC) [Wellman and Wurman, 1999] that is a testbed organised as an annual competition. In its classical version, agents are travel assistants that try to assemble the best travel package for their clients. To evaluate the assembling of packages they use the summation of the utilities of the clients that are in turn based on their preferences. It is a multi-issue negotiation process organised as a multi-lateral negotiation by means of auctions on every single issue. Bids take the form of pairs where a price per item and a number of items are proposed. There are also other versions of TAC problems that are similar to this: TAC/AA, CAT and TAC/SCM. TAC is a consolidated testbed, but its multi-lateral nature and mainly the absence of human agents makes it not appropriate for testing our agents.

Testbeds are often associated to competitions. That is the case of the above mentioned TAC, but also of the following testbed:

ART The Agent Reputation and Trust Testbed, [Fullam et al., 2005], was created with the aim of establishing a testbed for agent trust- and reputation-related technologies. Several competitions were organised where appraiser agents competed to get the higher reputation appraising paintings. The game is initialized with varying levels of expertise per agent in different artistic eras. Clients request appraisals for paintings from different eras. The agents should decide whether to request help from other appraisals paying an amount for information, or to deal with the appraisals by themselves. The higher the reputation of an agent, the easiest the other agents will accept to pay a lot.

The ART testbed is currently still very popular although the project is no longer maintained. It is a good testbed for trust and reputation. We think that trust is crucial for negotiation and thus, identifying what agents are trustworthy or not is necessary for any negotiating agent. Nevertheless, the ART testbed is too limited to trust and reputation.

CT The Colored Trails Game [Gal et al., 2005] is a research testbed for decision-making in groups comprising people and agents. In every game, two or more players may negotiate to exchange chips that allow them to move from their source position to a target position over a coloured squared board. To move through the board, a player must provide chips of the same colour of the squares that it wants to pass over. The settings of the game have a huge impact on the negotiation. Games may be made simple or complex along several dimensions including: number of players, size of the board, board information available to the different players, scoring rules for agents, the types of communication possible, and the negotiation protocol. Agents negotiate during specific communication phases exchanging messages that can be: chip exchange proposals, commits to proposals, retracts to previous committed proposals, and requests and suggests of plans towards the goal. The possible actions are: to move to an adjacent square investing a chip of the same colour, and to send chips to another player. The CT provides a graphical user interface for humans to take part in the game. When it is allowed by the specific settings of the given game, the interface provides a list of suggested paths to the goal. It is a nice interface that simplifies the work of the players making it easier for them to take part in the game.

CT is a simple but abstract game designed in a lab for research purposes. Humans taking part in the games need to be instructed beforehand. Monetary motivation is essential for congregating those people. The setting of the games affects a lot the negotiation and the outcome of the game. Special attention needs to be given in order to avoid unbalanced games with some players being initially in a better position than others. Contrarily, it is quite easy to create a software agent with the tools provided by the testbed.

GENIUS is a Generic Environment for Negotiation with Intelligent multi-purpose Usage Simulation that aims at facilitating the design of negotiation strategies, [Lin et al., 2011]. It focuses on bilateral negotiation and it is domain independent. It assumes that a given domain establishes a number of issues and a range of possible values per issue. The domain is known by all negotiating agents and fixed during a single negotiation session. A negotiation outcome is a mapping between issues and values. Every agent is completely aware of its own preferences and it is able to represent them as a normalised utility function. Preferences, and thus utility functions [Domshlak et al., 2011], are private and fixed. Interactions between negotiating parties are regulated by a negotiation protocol that is part of the setting of the negotiation session. GENIUS provides a repository of domains and utility functions as well as a toolbox for analysis that calculates optimal solutions and represents the evolution of the negotiation using it as a reference. This is possible thanks to assuming that preferences are known and fixed, that is something difficult to know in real world applications involving humans. In addition to the difficulties for humans to represent their preferences as an utility function, the required static fashion of those preferences is contradictory to the normal evolution of agent s internal state during negotiation interactions. Nevertheless, GENIUS provides a graphic user interface for human agents to negotiate among them or against software agents. With this interface, the environment can be used to teach strategic negotiation to humans.

GENIUS is known as a domain independent environment. Its authors state that negotiators must follow negotiation strategies that work well in several negotiation domains, not just one. However, they define a domain as a particular bilateral and multi-issue problem with fixed issues, fixed possible range of values per issue, and private but known fixed preferences. Despite of the variety of possible domains satisfying this requirements, they are all of the same kind. With respect to the use of GENIUS by humans, notice that it implies the teaching of humans to negotiate as software agents because they must be able to represent their preferences and keep them fixed during the whole negotiation session. Consequently, we think that using humans to evaluate agents in this environment has no sense. We are skeptical about the relevance of being able to teach humans to negotiate as software agents in GENIUS do. A good point of this testbed is that creating agents is simple, and that the testbed is very popular.

Since 2010, the Automated Negotiating Agent Competition (ANAC) has been celebrated annually on the context of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS) and the Workshop on Agent-based Complex Automated Negotiations (ACAN). It is now a popular competition among the automated negotiation research community. The purpose of the competition is to facilitate the research in the area of bilateral multi-issue closed negotiation, similar to what the TAC achieved for the trading agent problem. The idea is to compare software agents negotiating among themselves by pairs. To that end, the agents preferences and their associated utility functions are set by the competition organisers. ANAC is designed focused on the development of negotiation strategies rather than other important aspects of the negotiation. It uses GENIUS for the development and evaluation of agents, thus, its advantages and limitations are present in the competition. The competition sets an alternating-offers protocol in which negotiation parties exchange offers in turns. It uses several domains of different sizes and complexities. The competition represents linearly additive utility functions, so there are no dependencies between issues. This facilitates the modelling of the opponent s preferences. As the level of opposition of the utility functions affect the negotiation, those functions with a good variety of opposition are recommended allowing for a diversity of good outcomes. To come closer to realistic negotiation environments, deadlines and discount factors are used. Those factors affect the evaluation of the agents that are ranked at the end of the competition.

For the DipGame testbed, we rely on gamification and we use a well known and studied deterministic game whose features make it very trendy for the study of negotiation. The use of The Diplomacy Game as the domain for experimentation allows to get results of experiments involving humans playing against our software agents. Given the popularity of the game, users are motivated during the whole experiment. They use to feel identified by their role in the game and are enthusiastic of being able to play against software agents. There are several testbeds for negotiation described in this section, but none of them provides a comfortable environment for humans to take part of the experimentation. However, among all the testbeds described above, the CT is the most compelling to work with humans.

Basically, DipGame is more complex and requires more complete agents that, therefore, are more difficult to implement. The long duration of the game allows the establishment of relationships among the agents, and the study of the evolution of those relationships. The expressivity of the language is rich, allowing the exchange of information, emotions. It even allows to argue. Humans require testbeds like DipGame. Previous testbeds are not appropriate for bounded rational agents, that evolve, that are social, and that are motivated by their emotions.

2.4 The Diplomacy Game

The Diplomacy Game is a popular board game created by Allan B. Calhamer in 1954.[1] As Calhamer explains in [Calhamer, January 1974]:
"In 1959 I had 500 sets manufactured at my own expense after major companies had rejected the game. Manufacture was transferred to Games Research people incorporated in 1960. Sales have increased in every single year since the game has been on the market."

The game was commercialized by various companies over the years and was the first commercially published game to be played by mail. In 1983, the game started to be played by e-mail allowing quicker games. In 1988, the adjudication started to be done by computer. Adjudicating means to apply the rules of the game to change its state given the orders decided by the players. Two companies released in 1999 and 2005, a computer version of the game. Neither of them had good reviews as the computer AI was considered too poor. In the 21th century, people started playing online using web interfaces with automated adjudication. None of them included computer AI before DipGame that is described in Chapter 6.

In this section, we describe the game (Section 2.4.1), and the current community of players (Section 2.4.2).

2.4.1 Description of the game

In Diplomacy, players negotiate with the aim of conquering Europe. The rules of the game are unambiguous, and the available information about the game is rather rich as it is a quite old game being enjoyed by several generations of players: the game appeared in 1954. It is situated on Europe at the beginning of the 20^{th} century, before World War I. Each player is in charge of the armed forces, organised in *units*, of a major European power and must decide, in each turn, which movements the various units should execute. There is a maximum of one unit per province. The game ends when someone has an army powerful enough to control half of the special European `provinces called supply centres. This is achieved by defeating other players units, conquering their provinces and controlling an increasing number of supply centres that allow a player to get more units.

One of the most interesting features of Diplomacy is the absence of random movements: there are no cards and no dices. Also, this is not a turn-taking game. That is, all players move their units simultaneously: there is no advantage in a player being the first or last to move. All units are equally strong and

[1]Ranked 250 at `http://boardgamegeek.com/` by 6486 players.

consequently, when a unit attacks another, the winner of the battle is decided by taking only into account the number of units helping the fighting units. This feature is what makes Diplomacy so compelling for our purposes: the most relevant skills for a player are the negotiating ability, the intuition (knowing whom to trust) and the persuasive power.

The game splits in several years with two movement turns per year. In every movement turn, the players decide what movements their units should perform. Possible movements are: to move, to hold or to help another unit supporting its hold or move. When all players have decided their movements, those are made public at the same time and the game state is updated following the rules of the game.[2] The rules describe how to resolve conflicts that may emerge because of the concurrent announcement of movements. At the end of the year, if a player has a unit over a supply centre province, then the supply centre becomes owned by him/her. It will be him/her supply centre until a unit of another power conquers it at the end of a subsequent year period. At the end of every year, the number of units of all players are made equal to the number of owned supply centres by either building new units (when a player increased the number of owned supply centres) or removing existing units (when a player decreased the number of owned supply centres). Remember that owning half of the supplier centres of the Continent allows you to win the game. Therefore, the goal of the players is to increase the number of owned supply centres every year quicker than other players do.

All units always have the same strength. When there is an attack, it is resolved taking into account the number of supports that the attacking and defending units got from other units. Players can support the movements of other players units. In fact, these are the basics of the game: the players must convince other players to be their allies and to help them. This is done by negotiations that take place in parallel among all the players. In those negotiations, players ask others for help and sign deals on future plans of action like together attacking a unit of a player that they agree is a common enemy. From the point of view of a player, the most important aspect of the game is the negotiation process: deciding allies, selecting whom to ask for help, arguing with other players to get information about their objectives or to find out what they know, building trust and reputation, deciding when to honour a deal, maintaining relationships, and so on.

2.4.2 Player Community

Diplomacy is often played on the Internet. Interestingly, playing online makes easier to arrange long games with enough players and secretly to meet with other players to negotiate keeping conspiracies under wraps. There are several communities of Diplomacy players and most of them meet around any of the available options for playing the game. Maybe the oldest online community of players meets around *The Diplomatic Pounch*, [the]. People there uses *nJudge*, a software system developed in 1987 by Ken Lowe that allows players to play Diplomacy by e-mail and provides an automatic adjudicator.[3] This community

[2]Available at `http://www.wizards.com/default.asp?x=ah/prod/diplomacy`.

[3]The resolution of orders (i.e. actions performed by units) in Diplomacy is often called the adjudication process.

has an online magazine that contains articles analysing the strategy and tactics of the game [zin]. It is populated by many senior Diplomacy players and experts.

A frequent option is to play using a web application with a friendly interface for playing Diplomacy online. Playing online with a nice graphical interface facilitates the incorporation of newbies to the community spreading the game worldwide. There are basically two sites: *webDiplomacy* and *playDiplomacy*. Both sites allow playing online and sharing information with other members of the community. The community of *webDiplomacy* developed *phpDiplomacy* as an open-source project that makes internal use of *nJudge*. It seems that *playDiplomacy* took advantage of the source code of phpDiplomacy, and now it has the larger social network with over 50,000 members worldwide.

Other Diplomacy players invest their effort in developing software projects like *phpDiplomacy* or *jDip*. Some software applications try to help in either resolving the orders (adjudicators) or providing a nice interface for playing (mappers). An example is *jDip* that provide adjudication through a multi-platform desktop application [jDi, 2004]. This software is currently used to study tactics and analyse games. It is also an open-source project with a community of developers around. Its adjudication module and its map are used by other applications. The most popular community of Diplomacy developers is subscribed to the very active mailing list called *dipai* with more than 200 members. Some of the founders of that list developed the *Diplomacy Artificial Intelligence Development Environment* (DAIDE). DAIDE is a standard environment for the development of software programs, also software agents, to play The Diplomacy Game. The environment is quite mature, and there are several programs using the communication protocol [Rose, 2003] and message syntax [Norman, 2006].

A recent way of playing Diplomacy is using a mobile device. A company named Oort Cloud is powering Droidippy (http://droidippy.oort.se/); that is a platform that allows playing using both a web application and an Android application. Currently available games only involve human players, and have similar features to the previous mentioned web applications. Droidippy has also a forum.

Besides all the games played on the net using the mentioned software tools, a lot of tournaments of Diplomacy are being organised to play face-to-face games. The list of the best face-to-face players since 1977 from the most important tournaments is published at `http://en.wikipedia.org/wiki/International_prize_list_of_Diplomacy`. Those tournaments are mainly powered by the main Diplomacy Associations that are: *The North American Diplomacy Association*, *The European Diplomacy Association*, and *The Diplomacy Association of Australia and New Zealand*.

As one of the main contributions of this thesis, we designed a testbed based on The Diplomacy Game that makes use of DAIDE and jDip. Both are also used for applying our work to the game industry as described in Chapter 7. In addition, we collaborate with OOrt Cloud.

2.5 Agents playing Diplomacy

The idea of creating a Diplomacy software player (bot) emerged since the game was able to be played by e-mail. To our knowledge, the first bot to appear was *The Diplomat*, a bot created at Hebrew University in Israel. This bot is also called *The Israeli Diplomat* as later on some bot developers used the same name for their bots. *Diplomat* is another term to denote a bot.[4]

The majority of bots built to play The Diplomacy Game play a *no-press* version of it (i.e., without dialectic communication between players) [Loeb and Hall, 1990; Shapiro et al., 2002; Ritchie, 2003; Håård, 2004; Ribeiro et al., 2009]. The bots that are currently being developed run on DAIDE. Lamentably, not all of those bots are published. It is sometimes difficult to get documentation about them or their code. Even though, executable versions of them use to be available for running games in local competing against other bots or human players. As this thesis is related to negotiation, we describe only the works that provide some level of negotiation. Those are: *The Diplomat* [Kraus, 1995a; Kraus et al., 1989], *Shaheed's Diplomat* [Shaheed, 2004], *The Diplominator* [Webb et al., 2008], and *Stragotiator* [Kemmerling et al., 2009, 2011, 2012]. We describe them in the following after giving some details on Dumbbot, a very popular no-press bot that has been used as a starting point for building some of the existent press bots.

Dumbbot is quite popular because it is capable to beat newbie humans in spite of using a very simple algorithm. We consider it a good bot for training newbie players and teach them the rules of the game and simple tactics. It was created by David Norman, one of the founders and most active members of the dipai forum and developer of the infrastructure of DAIDE. The source code of Dumbbot is available at his own Diplomacy page `http://www.ellought.demon.co.uk/dipai`, and a good description of the bot is included in [de Jonge, 2010].

Dumbbot divides the map in regions[5] belonging to the Diplomacy provinces. Then, it assigns a *destination value* to some of them and spread that value towards the neighboring provinces with a decreasing factor. The idea is to provide paths towards the player s units should go in order to obtain a good outcome. Then, for each unit it decides the order that it should perform taking into account that destination value and the *competence* and *strength* of the order. The competence is computed given the number of units from other powers that are adjacent to a given region and the strength depends on the number of units that are supporting the unit moving or holding in the given region. Based on this, and using randomness to provide an indeterministic behaviour, it selects the moves to perform. It does not support convoys that is a type of move, and it has no negotiating capability.

Our agent s game strategy is based on Dumbbot s strategy. In addition, it distinguishes between not owned supply centres from proper provinces. And it enforces attacks towards not owned supply centres rather than other provinces. The competence and strength values are used only for negotiation. The spreading parameters are also adjusted.

The Diplomat was created between 1984 and 1989 during Sarit Krauss PhD thesis supervised by Daniel Lehman [Kraus, 1995a]. Eithan Ephrati and Arie

[4]We use the term *bot* instead of *AI*, *Diplomat*, *computer player character* or *agent* as it is nowadays the most used term and there could be a silly bot that is not using any artificial intelligence technique.

[5]He denotes regions by nodes, but it is the same meaning.

Schlesinger were MS students that assist this work providing game strategies and a message editor respectively. The Diplomat is a bot that consists of five modules implemented using several agents. Therefore, several agents were used to perform the work that corresponds to a human playing Diplomacy. [Kraus, 1995a] describes the structure of agents as a general automated negotiating agent evaluated in an environment that corresponds to The Diplomacy Game. The general structure can be described as a government with: a *Prime Minister*, a *Minister of Defense*, a *Foreign Office*, a *Military Headquarters*, and *Intelligence*.

The Prime Minister contains the personality traits of the bot that has the following parameters: aggressiveness, willingness to changes, and loyalty. It contains the bot knowledge and beliefs base in a module called Secretary. This information includes the rules of the game, the current state, a history of messages, an agreements table, and information about the other powers, and their relations, [Kraus et al., 1989]. The Ministry of defense is in charge of the planning and situation analysis using a module for Diplomacy strategy, [Ephrati, 1987]. It considers different fronts and coalitions that are sets of players that could be allied against others. The Foreign Office deals with the communication with other Diplomacy players. For that, they defined a Negotiation Language with: declarations, questions, suggestions and answers. Messages were translated either manually from natural English to the Negotiation Language, or written using a special editor. It establishes for each negotiation partner a Desk to deal with the bilateral negotiations with that partner. The Military Headquarters are responsible for decision making on moves, and the Intelligence models the other players behavior.

The structure assume two levels of distributed agents: a first level of autonomous selfish agents (some of them humans) that incarnate powers, and a second level of coordinated agents that work together as an automated selfish agent. Therefore, it is a MAS of coordinated agents incarnating a power in The Diplomacy Game that is, itself, another MAS of selfish agents. Those coordinated agents are local agents that use an internal-mail system for bilaterally communicating among themselves and a note board for broadcasting information to all the local agents. Although several agents are being used, they all run sequentially in the same processor. The priority for each agent (being a processing task) to be executed, and its processing state is regulated by the bot itself. The experiments were very limited and with no statistical significance. The source code is lost; there is no copy of it, and neither current machines would be able to run it. The best description of it is presumably included in the master thesis of Eithan [Ephrati, 1987] and Sarit Krauss PhD thesis [Krauss, 1988]; neither of them being able to be read by the author of this thesis.[6] This is a very valuable work because as its novelty and the state of technology in that decade. The Diplomat inspired the research of a lot of people included the author of this PhD thesis that discovered The Diplomacy Game thanks to it.

Jaspreet Shaheed's Diplomat is a bot that is also called Diplomat. It was build in 2004 as the master thesis of Jaspreet Shaheed. It runs on DAIDE and presents a novel approach to negotiation in Diplomacy using auctions, and simple algorithms to determine when to lie or not. They recognised units as resources, and thus, agreements on moves are allocation of resources.

[6]Although those works are available at Hebrew University, they are written in Hebrew.

They assumed a rational behaviour of the agents. Despite of that, they assume that other players strategies cannot be guessed. This Diplomat is structured in three components: strategy and tactics that generates the best possible plans[7], the negotiation finds agreements extending some of those plans including other agents moves, and deceipt that assess what agreements are beneficial if the agent lies and assess whether the other agents lie or not. The bot execution consists of three processes that are in charge of: (i) representing the bot properly; (ii) listening for messages from DAIDE server; and (iii) negotiating with the other players.

This bot uses a market based approach to negotiation consisting in biding units as they were resources [Shaheed, 2004]. Trust on other agents is assessed using Laplace s *Rule of Succession* based on the number of agreements kept given the total number of agreements. The bot has an *honesty* parameter that is used as a weight value when deciding whether to lie or not. The performance of the bot is tested against several no-press bots including Dumbbot. It outperforms a bot performing random moves and another that is only holding, but it plays worse than Dumbbot. Playing against no-press bots implies that the press bot cannot take advantage of its negotiation capability. Games with different number of Dumbbots and Diplomats were set, and the Diplomat at least outperformed Dumbbot in what concerns to avoiding being eliminated. The Diplomacy tactics of Dumbbot are better.

Stragotiator is a bot that realise a look-ahead planning. It is designed to be used within DAIDE. The aim of this work is to provide good negotiation capabilities in order to interact with other players and to appear mostly human , [Kemmerling et al., 2012]. According to them, a human-like behaviour corresponds to *The Classicist player* described by Windsor in [Windsor, 1999]. This is a loyal team-player that tries to maintain alliances welfare and to win with his allies. It does not break deals, either forgives betrayal. They implemented the Stragotiator as a Classicist player and approved its performance in a Turing test.

The Stragotiator main components are the strategy core, the negotiator and the opponent model. The opponent model is based on [Loeb and Hall, 1990] taking only into account the previous moves and giving a particular interpretation of them as intentions that increment positively or negatively a *friendship* model of the players relations. They also assume that the Stragotiator s strategy is the most intelligent and evaluate the other players *intelligence* as de difference between the decided moves and the moves that the Stragotiator would decide in that particular situation. *Trust* on the other players is also computed based on the execution of the previous deals. Positive and negative proposals do surprisingly also affect the trust. Trust is represented as a fuzzy set of four elements and defuzzification is done according to maximum membership.

The strategy core looks for positive partial joint move sets including moves for units of other players. A value is assigned to some province according to its status that is determined by its type and current use. Then the value is spread over the map to a maximum distance of four provinces similarly to Dumbbot. The bot s units try to move to the highest adjacent province or hold in its current province if this is the one with highest valued. Supports are preferred

[7]Those plans are sets of moves that do not include other agents' moves

than holds; thus the algorithm try to assign a support to each temporarily holding unit. Some provinces receive negative values meaning that the units must try to avoid them. Moving away when a unit is on a negative province.

An evolutionary algorithm is used to adjust several needed parameters like weights used to spread the value of the provinces, or the values that are initially assigned to provinces those are based on the province status. A uniform recombination is used with mutation of a move set by means of picking each unit with a given probability and changing its destination.

The negotiation is bilateral and allows for the exchange of move set proposals as well as peace or alliance proposals. It decides whether to accept or reject peace and alliances taking into account the opponent s model. After peace and alliances are accepted, the Stragotiator calculates moves for allied powers units using the strategy core. Based on those calculations, moves would be proposed. When receiving a move set proposal, it is accepted if the sender s trust is very high. Otherwise, the move set is analysed by the strategy core and accepted only if it is good enough.

The authors state that the Stragotiator is a believable Diplomacy bot and use their own measure argued in [Kemmerling et al., 2012] to evaluate it. They optimised the weighting values of it and published the source code at `http://www.irf.tu-dortmund.de/cms/en/Institute/Staff/Scientific_Staff/Kemmerling.html`. It is one of the last created negotiating bots.

2.6 Z-Notation

Z-Notation is a typed specification language based on set theory and first order predicate logic that is ideal for the specification of computer programs. It uses standard mathematical operations and provides a schema notation to allow the mathematical operations to be structured in a way that facilitates the posterior implementation of the computer programs that are specified with Z. A set of schema operations, most of them equivalent to mathematical operations, is also provided by Z to allow the combination of modules of equations [Bowen, 1996] building complex programs as the combination of several simpler modules. Z-Notation has been used to specify agent based systems before [d Inverno et al., 2004, 2012]. There are several versions of Z and tools like type checkers and syntax-aware editors. We concretely use Mike Spivey s second version of Z-Notation [Spivey, 1992] and a type-checker program called Fuzz [Spivey, 2008] to ensure that the specification is correct. A summary of the Z notation used can be found in Table 2.1.

In this thesis, we provide a formal specification of The Diplomacy Game using the Z-Notation formalism. We also include in it textual explanations of every single part of the specification to help understanding the specification.

Definitions and declarations

a, b	Identifiers
p, q	Predicates
s, t	Sequences
x, y	Expressions
A, B	Sets
R, S	Relations
$d; e$	Declarations
$a == x$	Abbreviated definition
$[A]$	Given set
$A ::= b\langle\!\langle B\rangle\!\rangle \mid c\langle\!\langle C\rangle\!\rangle$	Free type declaration
$\mathbf{let}\ a == x$	Local variable definition

Logic

$\neg\ p$	Logical negation
$p \land q$	Logical conjunction
$p \lor q$	Logical disjunction
$p \Rightarrow q$	Logical implication
$p \Leftrightarrow q$	Logical equivalence
$\forall X \bullet q$	Universal quantification
$\exists X \bullet q$	Existential quantification

Sets

$x \in y$	Set membership
$\{\}$	Empty set
$\mathbb{N}$	Set of Natural Numbers
$A \subseteq B$	Set inclusion
$\{x, y, \ldots\}$	Set of elements
$(x, y, \ldots)$	Ordered tuple
$A \times B \times \ldots$	Cartesian product
$\mathbb{P}\ A$	Power set
$\mathbb{P}_1\ A$	Non-empty power set
$A \cap B$	Set intersection
$A \cup B$	Set union
$A \setminus B$	Set difference
$\#A$	Size of a finite set
$\{d; e \ldots \mid p \bullet x\}$	Set comprehension
$A\ \mathsf{disjoint}\ B$	Disjoint sets
$s\ \mathsf{partition}\ A$	Partition

Functions

$A \pfun B$	Partial function
$A \longrightarrow B$	Total function
$A \twoheadrightarrow B$	Surjection

Relations

$A \longleftrightarrow B$	Relation
$\mathrm{dom}\ R$	Relation Domain
$\mathrm{ran}\ R$	Relation Range
$R^{\sim}$	Relational Inverse
$A \lhd R$	Domain restriction
$A \rhd R$	Range restriction
$A \ndres R$	Anti-domain restriction
$A \nrres R$	Anti-range restriction
$R \oplus S$	Relational overriding
R^*	Reflexive transitive closure
$R(\!\vert A\vert\!)$	Relational image
$R \fatsemi S$	Relational composition
$R \circ S$	Backward relational composition
$x \mapsto y$	Maplet

Sequences

$\mathrm{iseq}\ A$	Injective sequence
$\langle\rangle$	Empty
$\langle x, y, \ldots\rangle$	Sequence

Schema notation

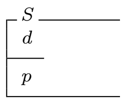
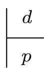
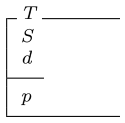
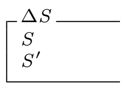
	Schema
	Axiomatic definition
	Schema Inclusion
	Operation
$a?$	Input to an operation
a	State component before operation
a'	State component after operation
S	State schema before operation
S'	State schema after operation
ΞS	No change of state

Table 2.1: Summary of Z notation used.

Chapter 3

Resource Negotiation Problem

In this chapter, we formalise the general problem that is the purpose of study in this thesis. The problem is denoted by Resource Negotiation Problem (RNP) and is a particular resource allocation problem where several agents control the resources of the environment and perform actions on it. Those actions guide the evolution of the environment. In Section 3.1, the environment is defined as well as the notion of plan and state. Section 3.2 contains the definition of the negotiation protocol of an RNP. Then, the problem is illustrated with an example in Section 3.3. The chapter ends in Section 3.4 with a summary.

3.1 Environment

We consider environments that are fully observable and regulated by a set of rules (physical or otherwise) that determine their evolution. Environments are populated by agents $\mathcal{A}$ that control resources R and are always in one of several possible states.

Definition 1 *Given a set $\mathcal{A}$ of agents and a set R of resources, an* environment state *$\omega \subseteq \mathcal{A} \times R$ is a set of agent-resource pairs. We denote by W the set of all possible environment states, that is $W = 2^{\mathcal{A} \times R}$*

$\langle \alpha, r \rangle \in \omega$ means that agent α controls resource r, and thus, it is the only agent that can act upon it.[1] We assume the existence of a finite set of operators Op that agents can apply to the resources they control. For instance, consuming the resource or transforming it. We thus define the set of possible actions as follows.

Definition 2 *The set of* actions *is the set $A = \mathcal{A} \times Op \times R$.*

We restrict the model to environments where no more than one operator can be applied to a resource simultaneously. This naturally leads to the definition of compatibility between actions.

[1]We will use the notation $\langle \cdot \rangle$ to denote elements of cartesian products.

Definition 3 *Two actions $a, b \in A$ such that $a = \langle \alpha, op_a, r_a \rangle$ and $b = \langle \beta, op_b, r_b \rangle$, are* compatible*, denoted by* comp(a, b)*, if and only if $op_a = op_b$ implies $r_a \neq r_b$.*

Controlling a resource means that only the agent that controls the resource can act upon it. This is our notion of action feasibility.

Definition 4 *An action $a = \langle \alpha, op, r \rangle \in A$ is* feasible *in state $\omega \in W$, denoted by* feasible(a, ω)*, if and only if $\langle \alpha, r \rangle \in \omega$.*

Actions are naturally grouped in sets, that we call plans, that without losing generality we can assume are executed at a given instant of time. Note that an agent can control more than one resource.

Definition 5 *A* plan *$p \subseteq A$ is a set of actions. The set of all possible plans is denoted by $P = 2^A$.*

We extend the notion of feasibility to plans in a natural way.

Definition 6 *Given a state $\omega \in W$ we say that plan $p \in P$ is feasible in state ω, denoted* feasible(p, ω)*, if and only if for all $a, b \in p$,* feasible(a, ω) *and* comp(a, b) *hold. The set of all feasible plans in state ω is denoted by P^ω.*

Two feasible plans are compatible if their actions are pair-wise compatible. That is, if its union is feasible.

Definition 7 *Given a state $\omega \in W$ and plans $p, q \in P$, we say that plans p and q are* compatible*, denoted* comp(p, q)*, if and only if their union is feasible, that is,* comp$(p, q) \Leftrightarrow p \cup q \in P^\omega$*.*

When an action is selected for each resource the plan is complete.

Definition 8 *Given a state $\omega \in W$ and a plan $p \in P$, we say that plan p is a* complete plan *for ω if and only if* feasible(ω, p) *holds and for all $\langle \alpha, r \rangle \in \omega$ then $\langle \alpha, op, r \rangle \in p$ for some $op \in Op$. We denote the set of all complete plans for state ω by $\bar{P}^\omega \subseteq P^\omega$ and by $\bar{P}^\omega_\alpha$ the projection of complete plans for α.*

Now we have all the ingredients to define environments as a type of deterministic transition system. That is, as a finite state machine with an initial state, with a set of final states, and with complete plans labeling the arcs between states.

Definition 9 *A* state transition system *is a tuple*

$$\Omega = \langle \mathcal{A}, R, Op, W, P, \mathbf{T}, \omega_0, W_f \rangle$$

where:

- *$\mathcal{A}$ is a set of agents*
- *R is a set of resources*
- *Op is a set of operators*
- *$W = 2^{\mathcal{A} \times R}$ is a set of states*

- $P = 2^{\mathcal{A} \times Op \times R}$ *is a set of plans*
- $\mathbf{T} : W \times P \to W$ *is a transition function such that* $\mathbf{T}(\omega, p)$ *is defined for all* $p \in \bar{P}^\omega$
- $\omega_0 \in W$ *is the initial state*
- $W_f \subseteq W$ *is the set of final states.*

The evolution of such a transition system is determined by a history of complete plans[2] being executed moving the state of the system away from the initial state, and eventually reaching a final state.

Definition 10 *Given a transition system* $\Omega = \langle \mathcal{A}, R, Op, W, P, \mathbf{T}, \omega_0, W_f \rangle$*, a* history *is a sequence of complete plans* $H = \langle p_0, p_1, \dots, p_n \rangle$ *such that for all* $0 < i < n$*,* $\mathbf{T}(\dots(\mathbf{T}(\omega_0, p_0), \dots), p_{i-1}) = \omega_i$ *and* $p_i \in \bar{P}^{\omega_i}$*. A history then implicitly defines an* environment state history *that is a sequence of states* $W_H = \langle \omega_0, \mathbf{T}(\omega_0, p_0), \mathbf{T}(\mathbf{T}(\omega_0, p_0), p_1), \dots \rangle$ *that we refer to as* $W_H = \langle \omega_0, \omega_1, \omega_2, \dots \rangle$*.*

3.2 Negotiation protocol

We define the negotiation in an RNP to be bilateral and satisfy a particular protocol. As an environment contains many agents, multiple bilateral negotiations can take place even simultaneously. The set of plans over which two agents negotiate is the set of feasible plans containing just actions performed by them. The plans involving up to two agents are called *negotiation options*, or options to simplify.

Definition 11 *A feasible plan* $\delta \in P^\omega$ *is called a* negotiation option *if and only if*

$$0 < \mid \{\alpha \in \mathcal{A} \mid \langle \alpha, op, r \rangle \in \delta\} \mid \leq 2$$

We denote by $\mathcal{O}^\omega \subseteq P^\omega$ *the set of negotiating options in state* ω *and by* $\mathcal{O}^\omega_{\alpha,\beta} \subseteq P^\omega$ *the negotiation options involving* α *and* β*.*

When two agents enact a negotiation protocol, they propose options that involve the two agents. Agents alternate on sending proposals and accepting or rejecting them until time expires. The possible messages exchanged betweewn two agents are called *negotiation utterances.*

Definition 12 *Given a transition system* $\Omega = \langle \mathcal{A}, R, Op, W, P, \mathbf{T}, \omega_0, W_f \rangle$*, a* negotiation utterance *in a state* $\omega \in W$ *is a tuple* $\mu = \langle \theta, \alpha, \beta, \delta \rangle$ *where* $\theta \in \{\text{propose}, \text{accept}, \text{reject}\}$*,* $\alpha \in \mathcal{A}$ *is the source,* $\beta \in \mathcal{A}$ *is the receiver of the utterance, and* $\delta \in \mathcal{O}^\omega_{\alpha,\beta}$*. We denote by* $M^\omega_{\alpha,\beta}$ *the set of all possible negotiation utterances between* α *and* β *in state* ω*.*

[2]Notice that in some application domains, there are default actions that do not need to be explicitly executed by agents as they are assumed. For example, in The Diplomacy Game (see Section 4.8) units are assumed to hold as default. Or in time tabling, agents are assumed to continue with the current scheduling of its events. The default actions are part of the problem definition and thus they are known by all the agents.

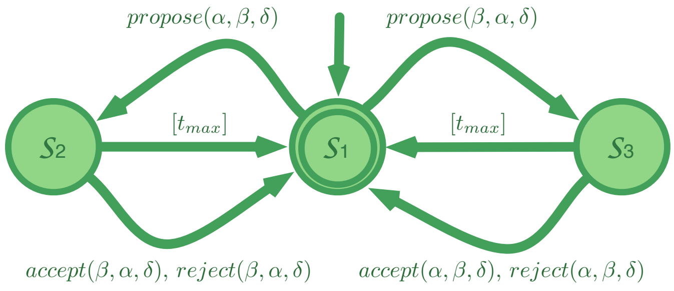

$propose(\alpha,\beta,\delta)$
$propose(\beta,\alpha,\delta)$
S_2
$[t_{max}]$
S_1
$[t_{max}]$
S_3
$accept(\beta,\alpha,\delta)$, $reject(\beta,\alpha,\delta)$
$accept(\alpha,\beta,\delta)$, $reject(\alpha,\beta,\delta)$

Figure 3.1: Negotiation protocol between two agents. Proposals are replied by accepts and rejects. It is not possible to send a proposal when a previous one is not yet replied. $[t_{max}]$ represents the end of the negotiation round.

In the following, we indistinctively represent utterances as tuples or predicates, e.g. $\langle propose, \alpha, \beta, \delta\rangle \equiv propose(\alpha, \beta, \delta)$. We define negotiation dialogues as sequences of utterances sorted by time.

Definition 13 *Given a transition system Ω, a* negotiation dialogue Ψ *in state ω between α and β is a sequence $\Psi = \langle \mu_0, \mu_1, \ldots, \mu_n\rangle$ such that $\mu_i \in M^{\omega}_{\alpha,\beta}$ for all $0 \leq i \leq n$.*

The negotiation protocol illustrated in Figure 6.12 determines what can be said and in which order. Dialogues are formed as utterance sequences so that each one is feasible with respect to the protocol. The next definition determines what utterances are feasible given a partial dialogue.

Definition 14 *Given a state $\omega \in W$, a dialogue $\Psi = \langle \mu_0, \mu_1, \ldots, \mu_{n-1}\rangle$ and an utterance $\mu_n = \langle \theta, \alpha, \beta, \delta\rangle$ we say that μ_n is* feasible *for dialogue Ψ, denoted by* feasible(Ψ, μ_n)*, if and only if:*

- $\Psi = \langle\rangle \Rightarrow \theta = \text{propose}$
- $\mu_{n-1} = \langle \text{propose}, \beta, \alpha, \delta\rangle \Rightarrow \theta \neq \text{propose}$
- $\mu_{n-1} \neq \langle \text{propose}, \_, \_, \_\rangle \Rightarrow \theta = \text{propose}$

The outcome of a successful negotiation that ended with an accept is a set of commitments on future actions to be performed by the negotiating agents. When an option being offered by agent α is accepted by agent β, it means that agents α and β commit to perform their actions in the option.

Definition 15 *Given a negotiation dialogue $\Psi = \langle \mu_0, \mu_1, \ldots, \mu_n\rangle$ we say that an action $a \in A$ is a* commitment *if $\mu_i = \langle accept, \_, \_, \delta\rangle \in \Psi$ and $a \in \delta$. We denote by C^{Ψ} the set of commitments in Ψ.*

3.3 Painting hanging example

In this section, we illustrate the use of RNPs by providing a simple example consisting on hanging a painting. This problem domain is not the most suitable to be studied as an RNP given the complexity of using humanoid robots to replace some of the human participants. Even though, we thought that it is the simplest and more broadly known example we can use. We refused to use Example 1 due to the huge diversity that exists of educational organisation timetables all around the world. Example 2 is a lot simpler, international and easy to understand.

Example 2 *Tonia wants to hang a heavy painting on a wall. To this aim, she requests the help of her sons Bep and Joan that will carry the painting. Tonia will be looking at the painting and giving advise on the correct position of it. At this moment, Bep and Joan are a meter away from the wall holding the painting and Tonia is more meters away in front of the painting looking at it.*

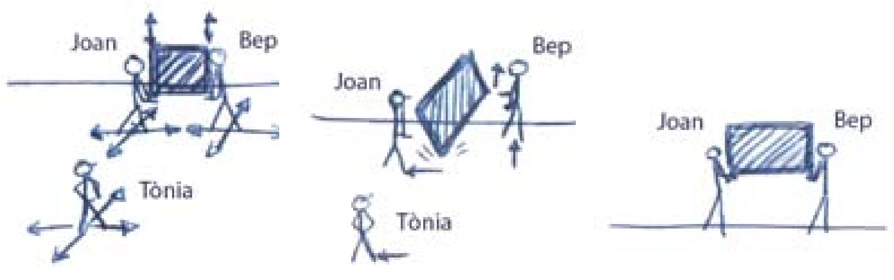


Figure 3.2: Painting hanging example.

Figure 3.2 illustrates the environment described in Example 2. It consists of three agents —Tònia, Bep and Joan, and a set of resources controlled by them. All agents have legs that can use to move around. Therefore, each agent has a resource that is its horizontal mobility, that has a value that is its horizontal position —a two dimensional space position. In addition, Bep and Joan have another resource that is the vertical mobility that can apply to their arms. They can rise and descend their arms. Contrarily, Tònia has no other resource than horizontal mobility because the other capabilities that Tònia, as a human, has are not used in her task, or are assumed to be used in a specific way that is invariant and known by all the agents. The total amount of resources is thus five.

The environment changes because of the agent's actions. The initial environment state is illustrated in Figure 3.2 a). There are several final states, those with the painting broken (see Figure 3.2 b)), and the one with the painting hanged in the correct position (see Figure 3.2 c)). At every non final state, agents decide what actions to perform, that is, what operator to apply to each of their resources. The set of operators that can be applied is limited, fully observable, and constrained by the environment. Thus, everybody knows at any time the resources controlled by every agent, and the operators that can be applied to them. In the painting hanging example, there are five operators that can be applied to the horizontal mobility resources: *keeping horizontal position*, *stepping forward*, *stepping backward*, *stepping left* and *stepping right.* And three operators applicable to vertical mobility resources: *keeping vertical*

Simbol	Action
a_0	Joan holds his horizontal position
a_1	Joan steps forward
a_2	Joan steps backward
a_3	Joan steps left
a_4	Joan steps right
a_5	Joan holds his vertical position
a_6	Joan rise his arms a bit
a_7	Joan descend his arms a bit
a_8	Bep holds his horizontal position
a_9	Bep steps forward
a_{10}	Bep steps backward
a_{11}	Bep steps left
a_{12}	Bep steps right
a_{13}	Bep holds his vertical position
a_{14}	Bep rise his arms a bit
a_{15}	Bep descend his arms a bit
a_{16}	Tònia holds his horizontal position
a_{17}	Tònia steps forward
a_{18}	Tònia steps backward
a_{19}	Tònia steps left
a_{20}	Tònia steps right

Table 3.1: Feasible actions.

position, *raising arms a bit*, and *descending arms a bit.* As Bep and Joan are carrying the painting, when they both raise their arms, they raise the painting. An example of action is Joan using his vertical mobility to *rise arms a bit.*

In general, we say that an action is feasible in an environment state if the resource being used is controlled by the agent acting on it. Table 3.1 lists the actions that are feasible in every non final environment state of the painting hanging RNP example. Then, two feasible actions are compatible if they concern operations on different resources. For instance, although Tònia strongly desire it, she cannot move her sons arms up and down. She can only request them to do that. Tònia rising Joan s arms is not a feasible action in any environment state of this RNP. It could be feasible in other RNPs where it were possible to reach an environment state with Joan s arms resource being controlled by Tònia. An example of two compatible actions is *Joan rising arms a bit* and *Joan stepping right.* He can do both things at the same time. Another example is *Bep stepping backward* and *Joan stepping backward.* These two actions are feasible and compatible, but undesirable the painting would fall and perhaps broke[3] (see Figure 3.2 b)).

We denote by a plan a set of actions to be performed. We say that a plan is feasible when its actions are feasible and pairwise compatible. The before

[3]Bep and Joan are looking at each other and they are holding the painting together: every one is holding one of the lower corners of the painting. Thus, the painting is situated between them. If both step backward, they step in opposite direction. The same happens when they both step right or left. If both try to step forward, their actions will not be executed because the painting is in between. In that case, they both remain in the same horizontal position.

examples of compatible actions do also form feasible plans containing only two actions. Two feasible plans are compatible when their union is a feasible plan. That is, if every action in the first plan is compatible with each action of the second plan. The plan containing the actions *Tònia keeping horizontal position*, *Joan stepping backward*, *Joan keeping vertical position*, *Bep stepping forward* and *Bep keeping vertical position* is a feasible plan. This feasible plan is also a complete plan as it has an action assigned to each resource. The projection for an agent, for instance Joan, is denoted by a complete plan for Joan. Thus, *Joan keeping horizontal position* and *Joan rising arms a bit* is a complete plan for Joan.

The environment evolves as the result of the execution of complete plans. The environment is regulated by a set of rules that determine its evolution given the actions executed by the agents. Rules can be physical as for instance gravity. An example of rule in the painting hanging environment is that the painting will fall if Joan and Bep both *step backward*, *step left* or *step right* at the same time, as the distance between them would be longer than the painting. The horizontal position of the agents in the environment states illustrated in Figure 3.2 is represented in Figure 3.3. We use this representation of the environment as it is simpler than the previous. Even thought, we should take into account that in order to reach the desired final state of Figure 3.2 c), Bep and Joan must have their arms raised in addition to having the horizontal position represented in Figure 3.3 c).

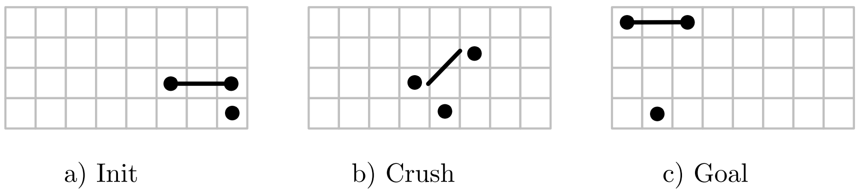


Figure 3.3: Horizontal positions. Points represent agents and the line is the painting. This figure corresponds to the horizontal positions of Figure 3.2.

In this simple example, the number of possible complete plans at each non final environment state is $5^3 \times 3^2 = 1125$.[4] The transition function $T(\omega, \bar{p})$ contains the physical rules that determine changes on the environment as the result of the execution of a given complete plan $\bar{p} \in \bar{P}^{\omega}$. In this example, T is quite obvious. It applies the mobility changes and the painting crashes only when the resulting positions imply a distance between Bep and Joan longer than the length of the painting. For example, when both step right or when the vertical position is not the same[5] (see Figure 3.2 b)).

The agents are autonomous. They can arbitrarily decide to move in any particular direction. We assume that agents do also have some sort of knowledge and motivations that drive their actions towards the achievement of their goals.

[4]Each one of the 3 agents can perform 5 different operations on its horizontal mobility resource, and 2 of them (Bep and Joan) can also perform 3 different operation on their vertical mobility resources.

[5]The painting is heavy and no one is capable to hold the entire weight of the painting by himself.

Thus, assuming that the agents know the correct place to hang the painting and that they all want it to happen, several paths can be followed in order to achieve the goal. In Figure 3.4, we illustrate some of the paths that Joan can follow towards the desired position. The achievement of the goals, and the path being used, will depend then on the decisions taken by the agents in each environment state.

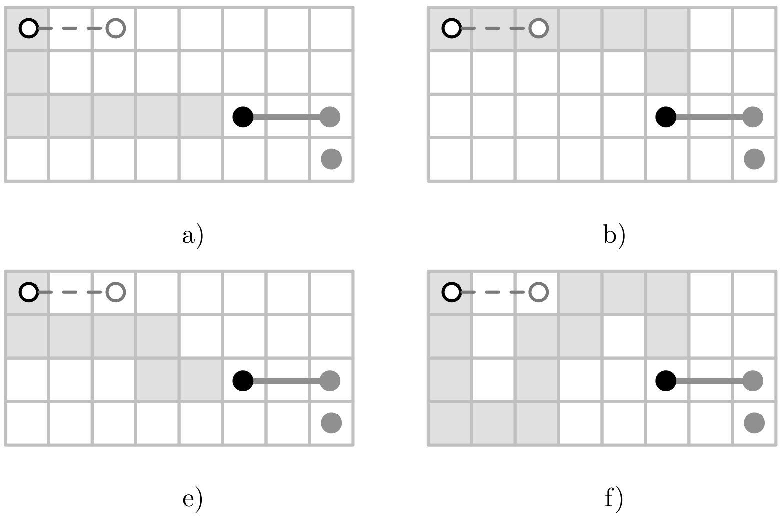


Figure 3.4: Some paths that Joan can follow towards the goal, Figure 3.3. Joan is the dark dot, the other agents are the grey dots and the painting is the strait line. Empty circles and the dotted line represent the correct positions. In light grey we represent some of Joan s possible paths.

In Figure 3.5, we represent the horizontal positions of some of the environment states of a hypothetical history $W_H = \langle \omega_0, \omega_1, \dots \omega_7 \rangle$. Given the environment state ω_2 the number of complete plans that allow the agents: to achieve the goal is 0,[6] to do not crash the painting is $5^2 \times 3 = 75$,[7] and to crash the painting is $5^3 \times 3^2 - 5^2 \times 3 = 1050$.[8] Thus, the probability of crashing is $1050/1125 = 93.\widehat{3}$ %. Even assuming that agents use the shortest paths to the goal, the number of complete plans that allow the agents: to achieve the goal is 0, to do not crash the painting is $2 \times 3 = 6$,[9] and to crash the painting is $2^2 \times 3^2 - 2 \times 3 = 30$. That corresponds to a probability of crash of $30/36 = 83.\widehat{3}$ %. That is too high for risking. Agents need to coordinate between themselves in order to achieve the correct position.

[6]It is not possible to achieve the goal from the current environment state. Agents can only approach to it.

[7]In order to protect the painting, the movements of one of the guys are determined by the movements of the other guy. The horizontal mobility of Tònia is independent to the painting being crashed.

[8]It is the set of remaining complete plans.

[9]There are several shortest paths to the goal and all of them require the holding agents to move in one of 2 directions. The horizontal movement of Tònia, assuming that she takes the shortest path is determined: she will step left. The vertical position is currently independent to the achievement of the goal.

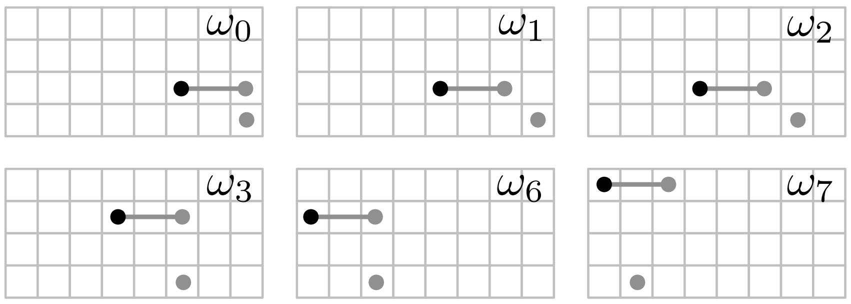


Figure 3.5: Environment states corresponding to the history W_H.

For the agents to be able to cooperate on, an RNP allows them to negotiate. Although in the painting hanging example the communication could be by broadcast, an RNP limits the communication to be bilateral. Thus, the messages exchanged between agents are private. Despite it sounds strange, when Tònia proposes to Bep to step right, Joan does not listen anything. He will receive only the messages sent by Tònia or Bep to him. Proposals can include more than an action, a plan. In spite of that, actions proposed can only refer to resources controlled by the two communicating agents. This special type of plan is denoted by negotiation option. Options can be proposed, accepted or rejected. For instance, Tònia would like to perform a joint plan where *Joan steps left*, *Bep steps right* and *Tònia steps forward.* This joint plan cannot be completely sent to any agent as it involves three agents. In order to negotiate the agreements on performing this plan Tònia would have to negotiate with Bep using options that are subsets of the desired joint plan. Tònia could propose to Bep the option including only the action *Bep steps left.* And then, propose to Joan the option including only the action *Joan step right.* Other options could be proposed including more actions.

For this toy example, the number of available options is small. The application domains that can be modeled with RNP use to be much more complex and provide, at any time, a huge number of possible plans and options. The negotiation protocol in RNP establishes that all proposals must be replied before proposing any other option to the same agent. Consequently, Tònia cannot propose anything else to Joan until she receives a reply from him.

The painting hanging example problem seems trivial, but it is not. It is small because the number of possible plans and rules is small. Even though, it requires a high level of coordination between the agents in order to fulfill Tònia s desire of hanging the painting.

3.4 Summary

A Resource Negotiation Problem (RNP) is a particular resource allocation problem with negotiating agents. An RNP is a MAS problem with several agents controlling the resources of a common environment. Agents individually act applying particular operators to their resources, and negotiate to agree on particular combinations of actions to be performed at the same time. Those combinations are denoted by plans. A plan contains a set of actions. A plan is complete if it contains an action for every resource in the environment. It is complete for a given agent if it contains an action for every resource in the environment controlled by that agent. An environment is a deterministic state transition system where the state is represented by the partition of resources among agents, and the transition function determines the next state given a complete plan. Thus, the evolution of the environment is given by the actions that agents execute.

The negotiation in RNPs is bilateral. The negotiation objects are denoted by negotiation options and are plans with actions involving only two agents. A negotiation utterance represents a message containing a negotiation option, a sender, a receiver, and an illocutionary particle that can be: propose, accept or reject. All agents controlling resources with actions included in the negotiation option must participate in the communication: either as sender or receiver. Therefore, it is not possible to negotiate about actions to be performed by a third party. The negotiation protocol establishes that all proposals must be replied. It enforces this forbidding to send a proposal to an agent that is waiting for our reply. Nevertheless, the agent is allowed to send a proposal to someone else. Therefore, the protocol permits concurrent negotiations among agents. When a negotiation option is accepted, the agents involved in the negotiation commit to perform the actions included in the negotiation option.

A simple example of an RNP is the problem of hanging a large and heavy painting. This task can be performed by three agents: an agent checking the correct position of the painting, and two agents carrying the painting. In this case, the resources that every agent control are their legs and arms. The final states of the state transition system representing this environment consist on crashing the painting that is an undesired final state, and having the painting hanged in the correct position that is the desired final state. The common goal of the agens is to reach the desired final state. To that end, the agents carrying the painting must be coordinated. And the third agent must inform about the position of the painting. Many possible sequences of complete plans can be executed to reach the goal. Many complete plans are available at each single environment state. Every agent may have their own criteria for choosing one of them; but they need to agree, on every single step, to avoid crashing the painting.

Chapter 4

The Diplomacy Game

The Diplomacy Game is a popular strategy board game played by seven players that perform moves concurrently, it is not a turn taking game. As a consequence of the concurrency of moves, conflicts emerge. The rules of the game establish a method for resolving those conflicts and proceeding generating the new game state for the next turn. This algorithm is denoted by the adjudicator algorithm. In Section 2.4, we described the game and its players community. We also explain that turns are identified by a year and a season. The names of the possible seasons are: movement, retreatement, and adjustment.

The specification included in this chapter corresponds to the standard version of The Diplomacy Game, but it is valid for every non-convoying[1] version of the game. The standard version uses a map of Europe similar to the political map of the Continent previous to the World War I. The game starts in 1901 and there are seven players incarnating the great powers of the decade that were intended to be: England, France, Italy, Germany, Austria, Russia and Turkey. During the game, players can negotiate in an unrestricted way and are not bounded by anything they say. The negotiation does not appear in the specification as there is no limitation on it and there is no control on whether or not commitments are honoured. The game is sometimes played with deadlines for turns and for the whole game that establish that the power with a better position to win when the deadline expires is the winner. Those deadlines are neither included in the standard version of Diplomacy and thus are not included in the specification.

In this chapter, we provide a formal specification of the game in Z-notation. Section 4.1 specifies the static part of the game, and Section 4.2 introduces the dynamic part. Then, the game state is formalized in Section 4.3. The initialization of the game is also described in this section. Next, the game proceeds applying the adjudicator algorithm to resolve conflicts and generate next game state. This algorithm is specified in parts corresponding to seasons: Section 4.4 describes the movement season, Section 4.5 describes the retreatement season, and Section 4.6 describes the adjustment season. Finally, we analyse the complexity of The Diplomacy Game in Section 4.7, and provide a representation of the game as a Resource Negotiation Problem in Section 4.8. The chapter concludes with a summary in Section 4.9.

[1]Convoys allow armies to browse the sea but we reject convoys, ignoring their rules, because they add unnecessary complexity to the game.

4.1 The Game

The game of Diplomacy situates each player in Europe at the beginning of the 20th century, before World War I. The board represents a map of that decade with some minimum variations, see Figure 4.1 (left). The map is composed by four different types of provinces that are *sea*, *coastal*, *inland* and *bi-coastal*. Figure 4.1 (right) illustrates those province types. The chips of the game represent military *units* that are either *armies* or *fleets*. Every unit is situated in the map hosted by a specific province. There is only one unit allowed per province at a time, see Figure 4.2 (top right), and the units that a province can host depend on the following limitations:

Armies: can be only in coastal, inland and bi-coastal provinces.

Fleets: can be only in sea, coastal and whatever coast of a bi-coastal province.

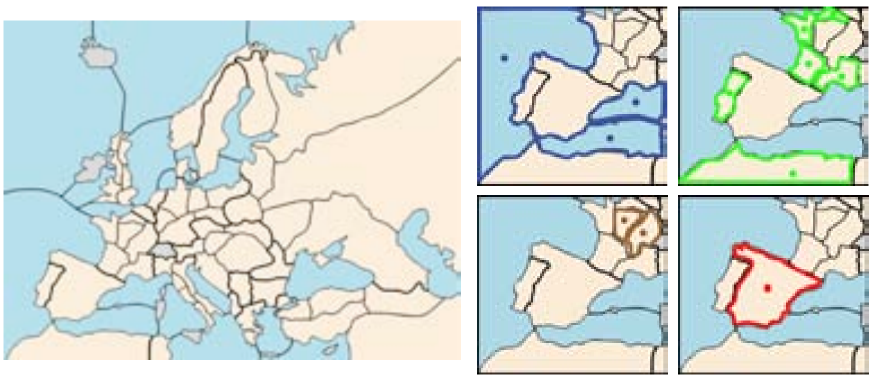


Figure 4.1: Full map (left) and province types (right) that listed from left right and top down are: sea, coastal, inland and bi-coastal provinces.

Notice that those limitations correspond to the natural fact that armies would sink into the sea and fleets cannot sail inland. Figure 4.2 (left) illustrates those limitations.

A unit can move from one province to another only if the provinces are adjacent, and both can host this type of unit. In the special case of a fleet moving to a bi-coastal province, the coast destination of the movement must be declared in advance. In fact, the fleet would only be able to move to a coast if it is adjacent along the coastline[2] to the province that is hosting it. Figure 4.2 (bottom right) illustrates the case of a fleet in Portugal that is a coastal province, moving to Spain that is a bi-coastal province. As Portugal is adjacent along the coastline to both coasts of Spain, the fleet can move to anyone of them. That is not always the case, sometimes there is only one coast adjacent along the coastline.

[2]Two costal provinces are adjacent along the coastline when their provinces are adjacent and there is a common adjacent sea province.

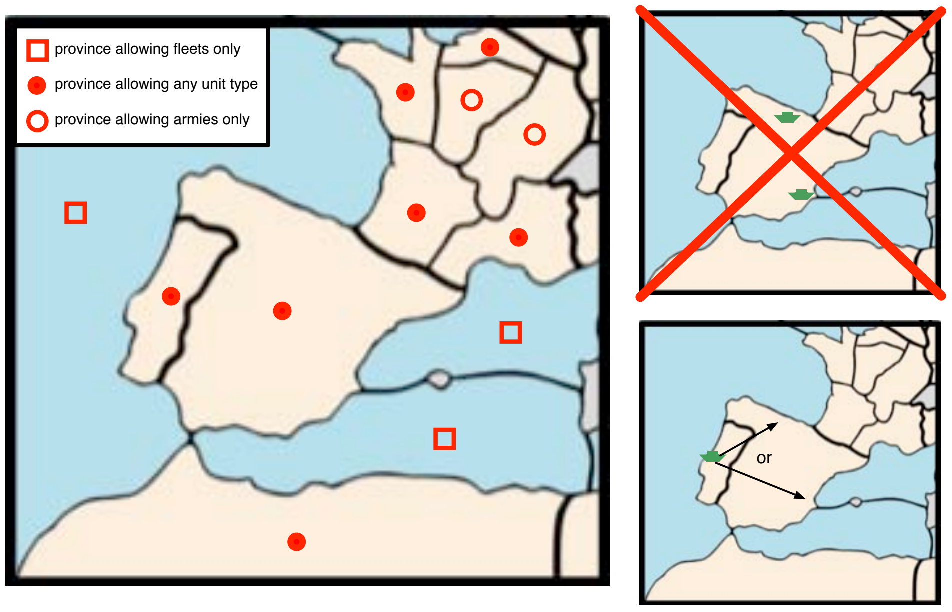

□ province allowing fleets only
● province allowing any unit type
○ province allowing armies only
or

Figure 4.2: Unit hosting limitation per province type (left). Any province can only host a unit (top right). Fleets moving to bi-coastal provinces should indicate the coast they want to go to (bottom right).

To simplify notation and avoid to constantly verify province and unit types, we introduce the abstract concept of *region*. We define a region as the capability of a province to host a unit. That is, every region belongs to a province, and somehow it represents the diversity of units that can be placed in the province, and its location into the province.[3] Figure 4.3 shows the correspondence between provinces and regions in a portion of the map. See there that, by introducing the concept of region we can say that:

sea and inland provinces: have only one region.

coastal provinces: have two regions, one for armies and another for fleets.

bicoastal provinces: have three regions, one for armies and the other two for fleets.

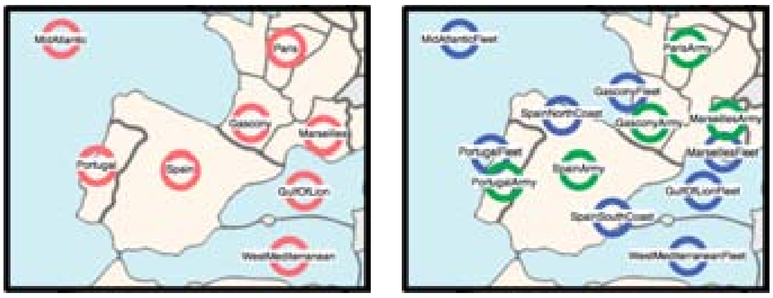
As can be seen in Figure 4.3 b), we label the regions with the name of their provinces followed by the word:

Army: if it can host an army.

Fleet: if it can host a fleet.

NorthCoast, EastCoast, SouthCoast or WestCoast: if it can host a fleet and it belongs to a bi-coastal province. This word specifies the coast.

[3]Remember that bi-coastal provinces allow fleets to be placed in one of their two coasts. The coast must be specified, Figure 4.2 (bottom right).



(a) Provinces (b) Regions

Figure 4.3: Example of correspondence between provinces and regions including their names.

For instance, the province of Portugal has two regions that are PortugalArmy and PortugalFleet while the province of Spain has three regions: SpainArmy, SpainNorthCoast and SpainSouthCoast.

Given the concept of region, we define the *adjacency relation* between two regions as the one that relates every two regions if (i) they can host the same type of unit; and (ii) they belong to adjacent provinces. See Figure 4.4 (left) for a graphical representation of the province and the region adjacency graphs. Then, the set of regions and the adjacency function form together a region adjacency graph with a special feature: the graph contains two disconnected subgraphs, one for fleets and another for armies. This feature guarantees that units moving between adjacent regions will never go to a region where it is not allowed to be placed just because there will be no path allowing it to happen. Figure 4.4 (right) illustrates the disconnected graph of regions for a portion of the map.

In Diplomacy, the players incarnate great powers. In the case of the standard version of the game, they are the following seven: France, England, Italy, Germany, Austria, Russia and Turkey. When playing, the identity of the players is often hidden under the powers, and it is only revealed when the game is over. From now on, we refer to the players as *powers*. Every power has a number of units under its *control*. Units have no identity. They only have a location and a controlling power. We refer to a unit as a region being controlled by a particular power. We define the type *POWER* as the set of the seven powers of the standard version, the type *REGION* as the set of all regions in the standard map of Diplomacy, and then we define the type *UNIT* as the cartesian product of powers and regions.

$$POWER ::= France \mid England \mid Italy \mid Germany \mid Austria \mid Russia \mid Turkey$$

$$REGION ::= SpainArmy \mid SpainNorthCoast \mid SpainSouthCoast \mid PortugalFleet \mid \ldots$$

$$UNIT == POWER \times REGION$$

Some of the provinces that are not sea provinces are *supply centres*. A supply centre is a special province that can be *owned* by a power. We define the type *SUPPLY CENTRE* as the set of all provinces in the map that are supply centres, and the type *GENERAL_PROVINCE* as the set of all provinces that

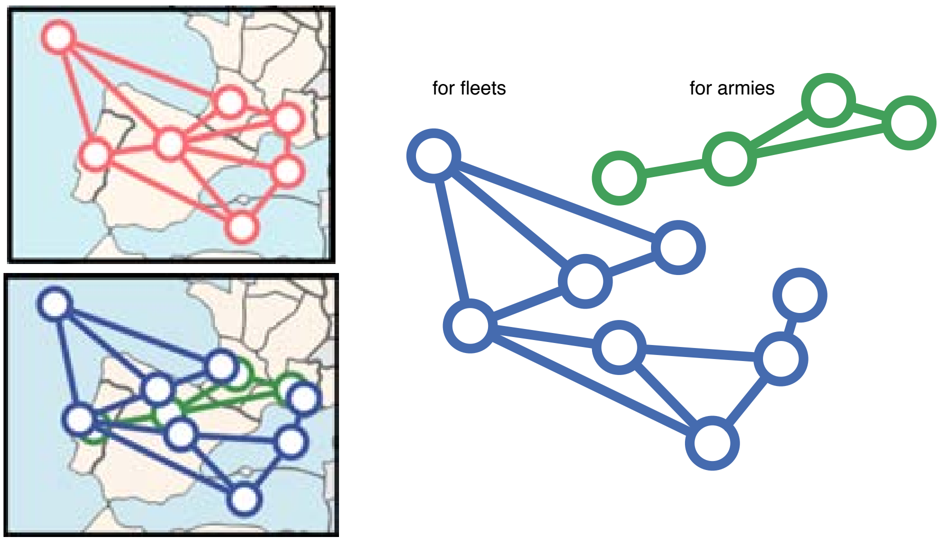


Figure 4.4: Adjacency graphs for a portion of the map. Province adjacency (top left), region adjacency (bottom left) and disjoint region adjacency graph (right).

are not supply centres. Then, we define the free type *PROVINCE* as the set of supply centres and general provinces. This free type declaration allows to use *sc* and *gp* to check whether a province is a supply centre or a general province.

$SUPPLY\_CENTRE ::= Spain \mid Portugal \mid Marseilles \mid Brest \mid ...$

$GENERAL\_PROVINCE ::= Gascony \mid Burgundy \mid York \mid ...$

$PROVINCE ::= sc\langle\langle SUPPLY\_CENTRE\rangle\rangle \mid gp\langle\langle GENERAL\_PROVINCE\rangle\rangle$

The game is divided in *years* and *seasons*. Years are natural numbers that are incremented one by one. The standard version of Diplomacy starts in 1901 and continues with 1902, 1903, ..., and so on until the end of the game. We define the type *YEAR* as the set of natural numbers that are greater or equal to 1901.

$YEAR == \{n : \mathbb{N} \mid n \geq 1901\}$

Every turn in a year receives the name of a season. The original rules of Diplomacy mention only two different seasons per year: spring and fall. Even though, paying attention to the rulebook content we can identify up to five different seasons per year: spring movements, spring retreats, fall movements, fall retreats and adjustments. We follow [Norman, 2006] and distinguish those five different seasons that we denote by (same order than previous five season list): *Spring*, *Summer*, *Fall*, *Autumn*, and *Winter*. Therefore, we define the type *SEASON* as the set of those five seasons.

$SEASON ::= Spring \mid Summer \mid Fall \mid Autumn \mid Winter$

Once types are defined, we continue defining the structure of the game that does not change through out the course of the game. This axiomatic definition includes the belonging relation between regions and provinces *province*, the adjacency relation of regions *adj*, the total number of supply centres *numOfSC*, and also the starting positions of the units over the map *startingUnits*. At the beginning of the game, some units are spread all over the map. They are initially hosted by supply centre provinces. Indeed, the set of supply centres where a power has units at the beginning of the game is denoted by the *homeland* of the power.

$$
\begin{array}{|l}
province : REGION \twoheadrightarrow PROVINCE \\
adj : REGION \leftrightarrow REGION \\
homeland : POWER \leftrightarrow SUPPLY\_CENTRE \\
startingUnits : POWER \leftrightarrow REGION \\
numberOfSC : \mathbb{N}_1 \\
\hline
\operatorname{dom} adj = REGION \\
adj^{\sim} = adj \\
\forall r : REGION \bullet \neg\ r \ \underline{adj}\ r \\
\forall ri, rj : REGION \mid ri \ \underline{adj}\ rj \bullet province(ri) \neq province(rj) \\
homeland = \{po : POWER; p : PROVINCE \mid \\
\quad po \mapsto p \in province \circ startingUnits \bullet po \mapsto sc^{\sim}(p)\} \\
\forall po : POWER \bullet \#(startingUnits(\!|\{po\}|\!)) < (numberOfSC \ \mathsf{div}\ 2)
\end{array}
$$

The constraints that must be always satisfied are (in the same order that they appear in the axiomatic definition):

1. Every region must be adjacent to another one. There is no isolated node in the adjacency graph.
2. The adjacency relation is symmetric.
3. The adjacency relation is anti-reflexive.
4. Two adjacent regions cannot belong to the same province.
5. A supply centre is homeland of a power if it has a starting unit on it.
6. All powers have less starting units than half the total number of supply centres.

The values of the previous variables depend on the Diplomacy version being played. The standard version of the Game assigns three starting units per power and four for Russia. Some units are armies and others are fleets. As can be seen in Figure 4.5, they are all hosted in regions belonging to the supply centre provinces that are historically related to those powers. In this way, the homeland of a power corresponds to the historical reality in Europe at the beginning of the 20th Century.

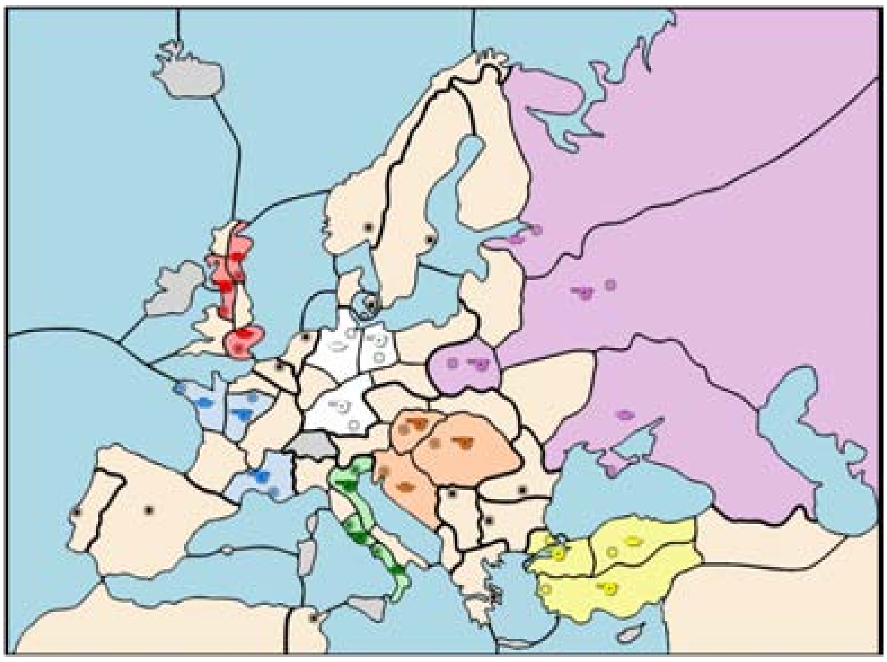


Figure 4.5: Board of a standard version of Diplomacy with the units in their starting positions.

4.2 The Game Dynamics

According to the rules of the game, all players reveal their moves at the same time. They incarnate great powers that decide what to order to their units to do. When they all have decided those orders, they make them public at the same time. Indeed, in Diplomacy, general moves are denoted by *orders* as they refer to what the powers ordering the units must do. There are several types of orders, among them *to move* and *to hold*. Using the term *order* we also avoid confusion between the specific order type *to move* and a general move that could be, in fact, *to hold*.

As introduced before, a year splits in five seasons. Those are of three different types: the movement, the retreatement, and the adjudication seasons. The movement seasons that are Spring and Fall, are used by players to move their units on the map. The retreatement seasons that are Summer and Autumn, let them retreat the units that were defeated and dislodged in the previous movement season, if there is any. Finally, the adjustment season that is Winter, is used to build and remove units whenever it is possible or needed. Therefore, every season type specify a different set of allowed order types.

The dynamics of the game establish the cycle of seasons: Spring, Summer, Fall, Autumn and Winter. That means that every year has two movement seasons followed by their respective retreatement season, and that the year ends with an adjustment season. To illustrate the season cycle we list the first turns of the standard version of Diplomacy: Spring 1901, Summer 1901, Fall 1901, Autumn 1901, Winter 1901, Spring 1902, Summer 1902, Fall 1902, Autumn

1902, Winter 1902 and Spring 1903. For each turn, the players decide what orders their units should perform and then the game state is updated to the next turn.

The players goal in a Diplomacy game is to take control of Europe. This is achieved when at least half of the supply centres belong to a player. The player that takes control of Europe wins and the game is over. Notice that only one player can win the game and that the powers goal will always be to own as many supply centres as possible.

A supply centre is owned by the last power that hosted a unit in it at the end of a year, that is in the adjustment season: Winter. Therefore, powers try to take control of the regions belonging to supply centre provinces and keep the unit in that position at least until the end of the year. As any supply centre is useful for a power, it is often the case of two powers wanting to own the same supply centre and moving their units towards it. This fact, together with the fact that moves are announced concurrently, promotes the emergence of *conflicts*.

A conflict takes place when more than one unit tries to be in the same province at the same time. This may happen either when more than one unit tries to move to the same province, or when one tries to hold in a province where others try to move towards. Conflicts represent battles between the conflicted units. The result of a battle can be as diverse as the battle itself. Sometimes the defeated unit gets dislodged and needs to be retreated in the next retreatement season.

Diplomacy is an easy game to play. The orders are quite simple and intuitive. The complex part of the game is the resolution of conflicts. The algorithm to resolve those conflicts and update the game state accordingly is called *the adjudicator algorithm*. This algorithm is normally executed by master players or computer programs. Many Diplomacy players do not completely understand the resolution process and, even though, they can play quite well. That is so because they focus only on the units that are near their units and pay more attention to the negotiation and deciding whether to trust others or not than to the rational probability of every possible order. In Section 4.7, we argue about this. To specify the adjudicator algorithm, we divide the algorithm in four parts: one for the game state and another one per season type. We describe those parts in the next sections 4.3, 4.4, 4.5 and 4.6. In those sections, several operations are defined as the combination of simpler ones. Finally, they are combined to form the adjudicator algorithm as described at the end of Section 4.6.

4.3 The Game State

A game consists of a sequence of turns. Those turns are identified by a year and a season. We represent the dynamic data of the game that corresponds to a particular turn with the *game state* concept. Thus, a game state contains among other data, its current year and season. We use a schema to formalise a game state. As you can see in the block below, the name of the schema is in the header (*GameState*), then some declarations are included in the top, and some constraints for them in the bottom. Those declarations correspond to the current year denoted by *currentYear*, the current season denoted by *currentSeason*, the relation of regions controlled by powers that represent the current units denoted by *control*, the relation containing the owned supply centres per power denoted by *own*, and the retreatement information denoted by *retreatement*. The retreatement information contains the relation of units dislodged in the previous turn and the possible unit destinations during a retreatement season. This means that the retreatement information will remain empty for turns that are not of the retreatement season type.

GameState

$$currentYear : YEAR$$
$$currentSeason : SEASON$$
$$control : POWER \leftrightarrow REGION$$
$$own : POWER \leftrightarrow SUPPLY\_CENTRE$$
$$retreatement : POWER \times REGION \leftrightarrow REGION$$

$$\forall ri, rj : REGION \mid ri \neq rj \land province(ri) = province(rj) \bullet \\ ri \in \operatorname{ran} control \Rightarrow rj \notin \operatorname{ran} control$$

$$\forall pi, pj : POWER \mid pi \neq pj \bullet \\ \mathsf{disjoint}\ \langle own(\!|\{pi\}|\!), own(\!|\{pj\}|\!)\rangle \land \\ \mathsf{disjoint}\ \langle control(\!|\{pi\}|\!), control(\!|\{pj\}|\!)\rangle$$

$$\forall p : POWER; r : REGION \mid (p, r) \in \operatorname{dom} retreatement \bullet \\ retreatement(\!|\{(p, r)\}|\!) \subseteq adj(\!|\{r\}|\!)$$

The constraints that a game state must satisfy are the following:

1. A province can only host a unit, meaning that only one of its regions can be controlled.

2. A supply centre cannot be own by more than a power at a time, and a region can neither be controlled by more than a power at a time.

3. A unit can only retreat to adjacent regions.

The game starts in Spring of the first year with the powers controlling the starting units and owning the homelands, the retreatement information is initially empty as there is no previous turn. Remember that in the standard version of Diplomacy, the first year is 1901.

The following schema denoted by *Init* is an operation that initialises the game. Operations in Z Notation are special schemas that can define changes, input and output variables. In this case, the operation declares in the top that it updates the *GameState* schema as indicated by the Δ symbol. The way to update the schema is included in the bottom. Dashed variables represent the final value of the variables once the schema is updated. Thus, the new value of *GameState* s *currentYear* variable would be always 1901 after executing the operation. Similarly happens with *currentSeason*, *control*, *own*, and *retreatement*.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Init}} \\ \hline
\Delta \textit{GameState} \\ \hline
\textit{currentYear}' = 1901 \\
\textit{currentSeason}' = \textit{Spring} \\
\textit{control}' = \textit{startingUnits} \\
\textit{own}' = \textit{homeland} \\
\textit{retreatement}' = \{\} \\ \hline
\end{array}
$$

4.4 Movements

The movement seasons are Spring and Fall. These seasons are used by powers to move their units all over the board. In fact, all units must receive an order that must be of one of the following types: (1) *hold* that is to remain in the same region; (2) *move* that is to leave the current position in order to stay in an adjacent region; (3) *support a hold order* that is to remain in the same position increasing the strength of another unit that is holding; and (4) *support a move order* that is to remain in the same position increasing the strength of another unit that is moving to a specific adjacent province.[4]

In the following schema denoted by *InputOrders* we define the type of orders that the players can perform in a movement season and the constraints that they must satisfy. At the top of the schema we see that it contains the *GameState*. Remember from previous definitions that the game state contains the current units represented by the *control* relation between powers and regions. Thus, given a game state we can refer to a unit by the region that is hosting it. Four input variables are also declared in the schema: *hold?*, *move?*, *supHold?*, and *supMove?*. They refer to the type of orders available in movement seasons. In Z Notation, input variables are followed by a question mark. To represent hold orders we use a set of regions containing all the regions hosting units that are ordered to hold. Move orders are represented by pairs of regions where the first element represents the unit that is ordered to move and the second element is the destination of the move. Similarly, supports to hold are pairs of regions where the first element represents the unit being ordered to perform the support and the second represents the unit that is receiving the support. Finally, supports to move are represented by 3-tuples where the first element is the region of the unit being ordered to perform the support, the second is the region of the unit receiving the support, and the third is the destination province of the supported move.

[4]Remember that convoys are excluded from this specification.

InputOrders

$$\begin{array}{l}
GameState \\
hold? : \mathbb{P}\ REGION \\
move? : REGION \mathrel{+\!\!\!\rightarrow\!\!\!\!\!\rightarrow} REGION \\
supHold? : REGION \mathrel{+\!\!\!\rightarrow\!\!\!\!\!\rightarrow} REGION \\
supMove? : REGION \mathrel{+\!\!\!\rightarrow\!\!\!\!\!\rightarrow} REGION \times PROVINCE \\
\hline
currentSeason = Spring \lor currentSeason = Fall \\
move? \subset adj \\
province \circ supHold? \subset province \circ adj \\
\mathrm{ran}\ supMove? \subset province \circ adj \\
\forall ri, rj : REGION; p : PROVINCE \mid supMove?(ri) = (rj, p) \bullet \\
\quad p \in (province \circ adj)(\!|\{ri\}|\!) \\
\langle hold?, \mathrm{dom}\ move?, \mathrm{dom}\ supHold?, \mathrm{dom}\ supMove? \rangle\ \mathsf{partition}\ \mathrm{ran}\ control
\end{array}$$

The input orders must satisfy the following constraints included in the bottom of the previous schema:

1. The current season must be a movement season.
2. A unit can only move to adjacent regions.
3. A unit can only support holding in neighbouring provinces.
4. A unit can only support correct movements.
5. A unit can only support movements towards neighbouring provinces.
6. Powers should assign only one order per controlled unit.

Conflicts appear often in movement seasons, its resolution is a bit complex and requires several steps to be undertaken. We define an operation for each of those steps and combine them at the end of this section to form the *Resolve-Movements* operation. To wrap the data that is common to those operations we use the *movements state* concept.

The *MovementsState* schema contains the *InputOrderes* and a set of variables used to keep track of the changes provided by the movements resolution steps. During those steps we asses the orders as either feasible or unfeasible. Feasible orders are successfully executed. Contrarily, unfeasible units are not. We use the partial function *assessedOrders* to be able to assign boolean values to the regions representing the units in order to assign them as feasible or unfeasible. The *feasible* and *unfeasible* sets of regions are also included as derived variables from the *assessedOrders* partial function.

Some unfeasible units may be dislodged as a consequence of a battle. Those are represented by the partial function *dislodged* that relates two pairs of regions where the first element represents the dislodging unit winner of the battle, and the second represents the dislodged unit. Sometimes, a battle consists of several units trying to move towards regions of the same empty province. In those cases if there is no unit winning the battle, the province remains standoff meaning that no unit can be retreated there in the next turn. Standoff provinces are represented by the set *standoff*. The orders being winning and defeated in a battle are decided depending on the strength of their orders. In the next section we compute the strength of all orders that is represented by the partial function *strength* that assigns a natural number to each region hosting a unit.

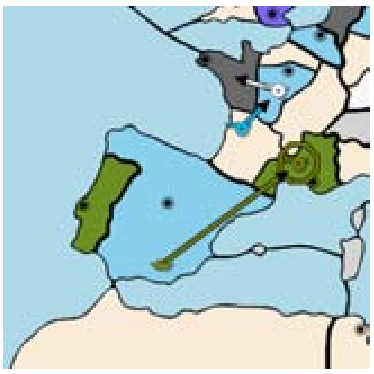
MovementsState

$$
\begin{array}{l}
\mathit{InputOrders} \\
\mathit{assessedOrders} : \mathit{REGION} \mathrel{+\!\!\!\!\rightarrow} \mathit{Bool} \\
\mathit{feasible}, \mathit{unfeasible} : \mathbb{P}\ \mathit{REGION} \\
\mathit{dislodged} : \mathit{REGION} \mathrel{+\!\!\!\!\rightarrow} \mathit{REGION} \\
\mathit{standoff} : \mathbb{P}\ \mathit{PROVINCE} \\
\mathit{strength} : \mathit{REGION} \mathrel{+\!\!\!\!\rightarrow} \mathbb{N} \\
\hline
\mathrm{dom}\ \mathit{assessedOrders} \subseteq \mathrm{ran}\ \mathit{control} \\
\mathit{feasible} = \mathrm{dom}(\mathit{assessedOrders} \rhd \{\mathit{True}\}) \\
\mathit{unfeasible} = \mathrm{dom}(\mathit{assessedOrders} \rhd \{\mathit{False}\}) \\
\mathrm{ran}\ \mathit{dislodged} \subseteq \mathrm{ran}\ \mathit{control} \\
\mathit{dislodged} \subseteq \mathit{move?} \\
\mathit{standoff} \subseteq \mathrm{ran}(\mathit{province} \circ \mathit{move?}) \\
\mathrm{dom}\ \mathit{strength} \subseteq \mathrm{ran}\ \mathit{control}
\end{array}
$$

A movement season state must verify the following constraints:

1. All assessed orders refer to controlled regions, that is to units.
2. The feasible regions are those that were assessed a true value.
3. The unfeasible regions are those that were assessed a false value.
4. Only controlled regions can be dislodged.
5. The dislodging units regions move towards dislodged units regions.
6. A standoff province is always the province destination of a move.
7. Only controlled regions can have a strength value.

The *MovementsState* schema contains input orders that do contain the game state. They both remain intact until the end of the movements resolution described in Section 4.4.4. The following Δ*MovementsState* operation protects the *GameState* and the *InputOrders* from being updated when updating the *MovementsState*. This is done by the use of the Ξ symbol.


(a) The green fleet performs a countryman attack, the blue army an attack and the white army a move.

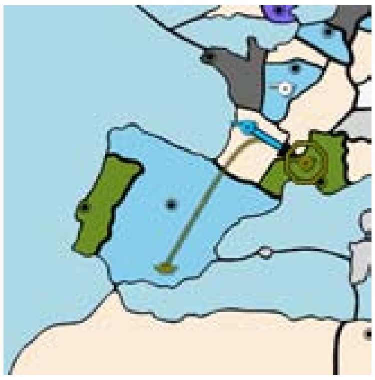

(b) The green fleet performs a countryman attack support to the blue army towards the green army.

Figure 4.6: Example of moves attacks and countryman attacks and supports.

$\Delta \mathit{MovementsState}$

$\mathit{MovementsState}$

$\mathit{MovementsState}'$

$\Xi \mathit{GameState}$

$\Xi \mathit{InputOrders}$

4.4.1 Order strength

Conflicts between orders in movement seasons represent battles. The outcome of those battles basically depend on the strength of the orders that are in conflict. The strength of an order is computed from feasible supports. The more feasible supports received by an order the more strength it has.

Moves towards non empty destinations are called attacks. Figure 4.6 a) show two moves. The one performed by the army is a simple move. The one performed by the fleet is an attack as it is moving towards a province hosting another unit. The destination of an attack is called a target.

When the attacked unit is controlled by the power that controls the attacking unit, the attack is called a *countryman attack*. This is also the case of the fleet in 4.6 a). Countryman attacks cannot dislodge their targets. Similarly happen when the unit that is supporting an attack is controlled by the same power that is controlling the unit that is being attacked. This is the case illustrated in 4.6 b). The supporting unit is performing a countryman attack support and, thus, the support order is unfeasible. It does not matter who is the power controlling the attacking unit in order to say that there is a countryman attack support.

By default, all supports are feasible. For a support to be unfeasible and thus, be assessed as false, it must violate one of the following rules:

1. A unit cannot support a second unit to hold when the second unit is moving.

2. A unit cannot support a move that is not ordered.

3. A power cannot support an attack against itself. Countryman attack supports are unfeasible.

4. A first unit cannot support a second one to hold when the first unit is being attacked.

5. A first unit cannot support a second one to move when the first unit is being attacked.

The *DetectUnfeasibleSupports* operation checks the previous rules for all support orders and assesses to false all supports that do not satisfy those rules. The rest of the variables of the *MovementsState* remain the same.

$$
\begin{array}{|l}
\textit{DetectUnfeasibleSupports} \\
\hline
\Delta \textit{MovementsState} \\
\hline
\textit{assessedOrders}' = \\
\quad \{ri, rj : \textit{REGION} \mid ri \mapsto rj \in \textit{supHold}? \land rj \in \operatorname{dom} \textit{move}? \bullet \\
\quad\quad ri \mapsto \textit{False}\} \\
\quad \cup \\
\quad \{ri, rj : \textit{REGION}; p : \textit{PROVINCE} \mid ri \mapsto (rj, p) \in \textit{supMove}? \land \\
\quad\quad rj \mapsto p \notin (\textit{province} \circ \textit{move}?) \bullet \\
\quad\quad\quad ri \mapsto \textit{False}\} \\
\quad \cup \\
\quad \{ri, rj : \textit{REGION}; p : \textit{PROVINCE} \mid ri \mapsto (rj, p) \in \textit{supMove}? \land \\
\quad\quad (\textit{province} \circ \textit{control})^{\sim} (\!| \{p\} |\!) = \textit{control}^{\sim} (\!| \{ri\} |\!) \bullet \\
\quad\quad\quad ri \mapsto \textit{False}\} \\
\quad \cup \\
\quad \{ri, rj, rk : \textit{REGION} \mid ri \mapsto rj \in \textit{supHold}? \land rk \mapsto ri \in \textit{move}? \land \\
\quad\quad \textit{control}^{\sim} (\!| \{ri\} |\!) \neq \textit{control}^{\sim} (\!| \{rk\} |\!) \bullet \\
\quad\quad\quad ri \mapsto \textit{False}\} \\
\quad \cup \\
\quad \{ri, rj, rk : \textit{REGION}; p : \textit{PROVINCE} \mid ri \mapsto (rj, p) \in \textit{supMove}? \land \\
\quad\quad rk \neq rj \land rk \mapsto ri \in \textit{move}? \land \textit{control}^{\sim} (\!| \{rk\} |\!) \neq \textit{control}^{\sim} (\!| \{ri\} |\!) \bullet \\
\quad\quad\quad ri \mapsto \textit{False}\} \\
\textit{standoff}' = \textit{standoff} \\
\textit{dislodged}' = \textit{dislodged} \\
\textit{strength}' = \textit{strength} \\
\hline
\end{array}
$$

The next operation denoted by *SetFesibleSupports* assesses to feasible all the remaining support orders as they satisfy the previous rules.

SetFesibleSupports

$\Delta MovementsState$

$$
\begin{aligned}
& assessedOrders' = assessedOrders \cup \{r : REGION \mid \\
& \quad r \notin \operatorname{dom} assessedOrders \wedge \\
& \quad (r \in \operatorname{dom} supHold? \vee r \in \operatorname{dom} supMove?) \\
& \quad \bullet\ r \mapsto True\}
\end{aligned}
$$

$standoff' = standoff$

$dislodged' = dislodged$

$strength' = strength$

Finally, we compute the *strength* of each order taking only into account the feasible supports. The strength of an order is equal to the number of feasible supports that it receives plus one. The operation *ComputeStrength* updates the strength function assigning to each controlled region a value computed as follows:

- The controlled regions that are not ordered to move have a strength equal to 1 plus the number of feasible supports to hold that they receive. That is, the strength of region r is 1 plus the number of regions ri belonging to *feasible* such that the pair $ri \mapsto r$ belongs to *supHold?*. In short this is:

  $1 + \#(feasible \lhd supHold? \rhd \{r\})$

- The controlled regions that are ordered to move have a strength equal to 1 plus the number of feasible supports to move that they receive. That is, the strength of region r that is being ordered to move to a region belonging to province pr is 1 plus the number of regions ri belonging to *feasible* such that $ri \mapsto (r, pr)$ belongs to *supMov?*. And in short:

  $1 + \#(feasible \lhd supMove? \rhd \{(r, province(move?(r)))\})$

ComputeStrength

$\Delta MovementsState$

$$
\begin{aligned}
& strength' = \\
& \quad \{r : REGION \mid r \in \operatorname{ran} control \setminus \operatorname{dom} move? \bullet \\
& \quad r \mapsto 1 + \#(feasible \lhd supHold? \rhd \{r\})\} \cup \\
& \quad \{r : REGION \mid r \in \operatorname{dom} move? \bullet \\
& \quad r \mapsto 1 + \#(feasible \lhd supMove? \rhd \{(r, province(move?(r)))\})\}
\end{aligned}
$$

$standoff' = standoff$

$dislodged' = dislodged$

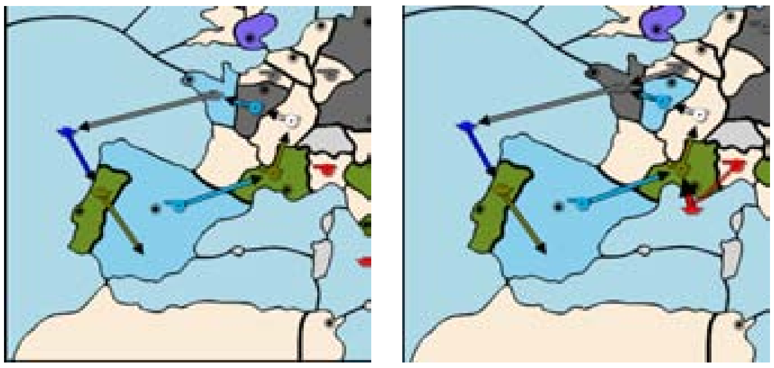
$assessedOrders' = assessedOrders$

We conclude this section combining the previous operations as the following sequence of operations denoted by *ComputeOrderStrength*:

$ComputeOrderStrength \mathrel{\widehat{=}} DetectUnfeasibleSupports \mathbin{⨾} SetFeasibleSupports \mathbin{⨾} ComputeStrength$

4.4.2 Avoiding loops

This step of the movements resolution consists in avoiding loops that could appear resolving battles. This is done by detecting cycles of attacking units. Those cycles are called *rings of attack* when there is no external attack breaking the ring. Otherwise we call it a broken ring. A cycle consists of a sequence of units attacking each one to the next, and the last unit attacking the first. Figure 4.7 a) represents a ring of attack. When the cycle contains only two units, it is called a *head to head battle* instead of a ring.



(a) The moving units are performing a ring or attack. There is no external attack as strong as the internal attacks.

(b) This ring is broken by the grey army from the north and by the red fleet and the red army from the south.

Figure 4.7: Example of cycles of attack including a ring and a broken ring.

An external attack is an attack performed by a unit that is not included in the cycle but that it is moving towards a region hosting one of the units forming part of the cycle. Similarly we denote by internal attacks those attacks performed by the units of the cycle. If there is at least a unit receiving an external non countryman attack at least as strong as the internal attack, then the ring is broken and the loop does not indeed exist. Figure 4.7 b) represents a broken ring.

The next operation *ResolveRings* define the local variable R and assigns to it the set of cycles that are rings of attack. In Z Notation, a local variable is declared using the word **let** . Injective sequences are declared using the word iseq. We define a ring of attack as an injective sequence of regions where every region attacks the next region in the sequence and the last region attacks the first one. $\#\sigma$ is used to get the number of elements in the sequence σ and mod to compute the arithmetic modulus.

$\sigma : \mathrm{iseq}\, REGION \mid (\forall\, i : \mathbb{N};\ rk : REGION \mid i \geq 1 \land i \leq \#\sigma \bullet$
$move?(\sigma(i)) = \sigma(((i+1) \mathsf{\ mod\ } \#\sigma) + 1)$

For the cycle to be a ring of attack, it must also verify that for every region rk representing an external attack, the strength of this attack must be lower than the strength of the internal attacker.

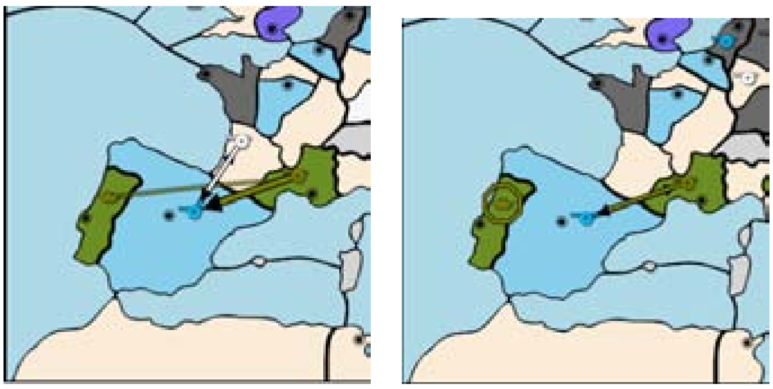
$rk \notin \mathrm{ran}\, \sigma \land move?(rk) = \sigma(((i+1) \mathsf{\ mod\ } \#\sigma) + 1) \land$
$strength(\sigma(i)) > strength(rk)) \land$

And the number of elements of the sequence must be greater than 2: $\#\sigma > 2$

All regions involved in a ring of attack are assessed as feasible and their external attackers are assessed as unfeasible.

```
ResolveRings
  $\Delta MovementsState$
  ---
  $(\textbf{let } R == \{\sigma : \text{iseq}\, REGION \mid (\forall i : \mathbb{N};\ rk : REGION \mid i \geq 1 \wedge i \leq \#\sigma \bullet$
    $move?(\sigma(i)) = \sigma(((i+1) \bmod \#\sigma) + 1) \wedge$
    $rk \notin \text{ran}\, \sigma \wedge move?(rk) = \sigma(((i+1) \bmod \#\sigma) + 1) \wedge$
      $strength(\sigma(i)) > strength(rk)) \wedge \#\sigma > 2 \bullet \sigma\} \bullet$
    $assessedOrders' = assessedOrders \oplus$
      $\{rj : REGION;\ \sigma : \text{iseq}\, REGION \mid$
        $\sigma \in R \wedge rj \in \text{ran}\, \sigma \bullet rj \mapsto True\}$
      $\cup$
      $\{rj : REGION;\ \sigma : \text{iseq}\, REGION \mid$
        $\sigma \in R \wedge rj \notin \text{ran}\, \sigma \wedge move?(rj) \in \text{ran}\, \sigma \bullet rj \mapsto False\})$
  $standoff' = standoff$
  $dislodged' = dislodged$
  $strength' = strength$
```

A head to head battle consists of two units attacking each other. The resolution of a head to head battle depends on whether it is balanced or unbalanced. It is balanced if both units have the same strength to move or if they perform countryman attacks, meaning that both units are controlled by the same power. Otherwise, the battle is unbalanced. Figure 4.8 illustrates both types of head to head battles.


(a) Head to head battle between the blue and the white armies that is balanced by the green attack.

(b) Unbalanced head to head battle. Both attacks are equally strong and no external attacker found.

Figure 4.8: Example of head to head battles.

We represent the set of head to head battles as the symmetric relation of regions hosting a pair of units that take part of a balanced head to head battle. They have either the same strength or are controlled by the same power. We assign that relation to the local variable B:

$B == \{ri, rj : REGION \mid \{ri \mapsto rj, rj \mapsto ri\} \subseteq move? \wedge$
$(strength(ri) = strength(rj) \vee control^{\sim}(\!|\{ri\}|\!) = control^{\sim}(\!|\{rj\}|\!)) \bullet ri \mapsto rj\}$

The *ResolveBBattles* operation detects balanced head to head battles, assess them to unfeasible and looks for a single strongest external attacker. If there is only one external attack with the highest strength and it is strongest than the internal attack, then this single strongest external attack is assessed to feasible.

```
ResolveBBattles
  $\Delta MovementsState$
  ----
  $(\mathbf{let}\ B == \{ri, rj : REGION \mid \{ri \mapsto rj, rj \mapsto ri\} \subseteq move? \wedge$
   $(strength(ri) = strength(rj) \vee control^{\sim}(\!|\{ri\}|\!) = control^{\sim}(\!|\{rj\}|\!)) \bullet ri \mapsto rj\} \bullet$
    $assessedOrders' = assessedOrders \oplus$
     $\{r : REGION \mid r \in \mathrm{dom}\, B \bullet r \mapsto False\}$
     $\cup$
     $\{ri, rj : REGION \mid rj \mapsto ri \in move? \wedge ri \in \mathrm{dom}\, B \wedge rj \notin \mathrm{dom}\, B \wedge$
      $(\forall rk : REGION \mid rk \neq rj \wedge rk \mapsto ri \in move? \bullet$
      $strength(rk) < strength(rj)) \bullet rj \mapsto True\})$
  $standoff' = standoff$
  $dislodged' = dislodged$
  $strength' = strength$
```

The *SetDislodgements* operation detects the set of balanced head to head battles and assesses as unfeasible all regions that are attacking a unit in a balanced head to head battle that are not assessed as feasible. Thus, only the strongest external attacker, if there is any, will be feasible. This operation also sets as *dislodged* all regions whose units took part of a balanced head to head battle and that are receiving a feasible attack; the attack is indeed performed by a strongest external attacker.

```
SetDislodgements
  $\Delta MovementsState$
  ----
  $(\mathbf{let}\ B == \{ri, rj : REGION \mid \{ri \mapsto rj, rj \mapsto ri\} \subseteq move? \wedge$
   $(strength(ri) = strength(rj) \vee control^{\sim}(\!|\{ri\}|\!) = control^{\sim}(\!|\{rj\}|\!)) \bullet ri \mapsto rj\} \bullet$
    $assessedOrders' = assessedOrders \oplus \{ri, rj : REGION \mid ri \in \mathrm{dom}\, B \wedge$
      $rj \mapsto ri \in move? \wedge rj \mapsto True \notin assessedOrders \bullet rj \mapsto False\} \wedge$
    $dislodged' = dislodged \cup \{ri, rj : REGION \mid rj \in \mathrm{dom}\, B \wedge$
      $ri \in \mathrm{dom}(feasible \lhd move? \rhd \{rj\}) \bullet ri \mapsto rj\})$
  $standoff' = standoff$
  $strength' = strength$
```

Resolving an unbalanced head to head battle consists simply in assessing the weaker unit of the battle to unfeasible. This way the loop is broken and the battle can be resolved in the ordinary way that is described in next section. *ResolveUBattles* operation detects unbalanced head to head battles, identifies the attack with less strength and assesses it to unfeasible. Remember that for the battle to be unbalanced, an attack must be stronger than the other and the units must be controlled by different powers. If they performed countryman attacks, the battle would be balanced.

ResolveUBattles

$\Delta MovementsState$

$assessedOrders' = assessedOrders \oplus \{ri, rj : REGION \mid$
$\{ri \mapsto rj, rj \mapsto ri\} \subseteq move? \land strength(ri) > strength(rj) \land$
$control^\sim (\!| \{ri\} |\!) \neq control^\sim (\!| \{rj\} |\!) \bullet rj \mapsto False\}$

$standoff' = standoff$

$dislodged' = dislodged$

$strength' = strength$

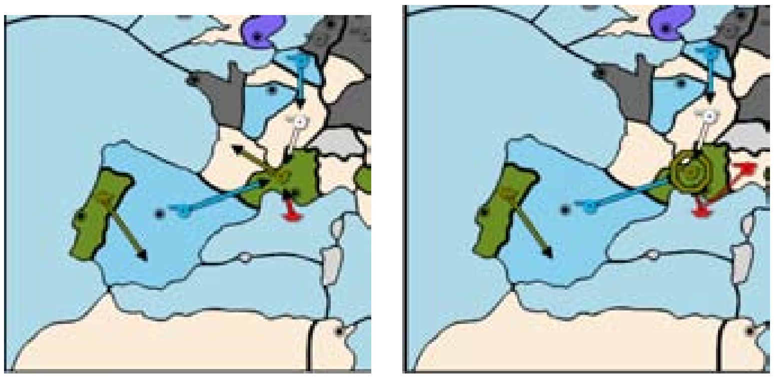
Finally, we define the *AvoidLoops* operation as the sequence of all the previous operations defined in this section:
$AvoidLoops \mathrel{\widehat{=}} ResolveRings \fatsemi ResolveBBatles \fatsemi SetDislodgements \fatsemi ResolveUBattles$

4.4.3 Battles

Once loops are avoided we can proceed resolving the ordinary battles iteratively knowing that the process will reach an end. Ordinary battles are those conflicts that are still not resolved. Those ordinary battles can be simple moves towards empty regions or attacks towards occupied regions. In both cases, the region is denoted by the *target*. If the destination of the move is empty, the single strongest move towards that target is feasible and the rest are unfeasible. Notice that there can only be one strongest move. When several moves that are equally strong are the strongest, all the moves are unfeasible. As a consequence of the battle involving all those moves towards the target, the province of the target is set to standoff. This means that, although the province will remain empty, it cannot be used in next turn as the destination of a retreatement.

When the target is occupied, we first check whether there is a single strongest attacker. Again, the strongest attacker towards tha target must be unique. Otherwise all attacks are unfeasible. When there is a single strongest attacker, the rest of attacks are set as unfeasible and the strength of the strongest is compared to the strength of the defender, that is the unit occupying the target. The defender is defeated only when the attacker is stronger than it. Otherwise, the attack is unfeasible. When the strongest is stronger than the defence, it is set as feasible and the defender is dislodged. That means that the attacker s new position will be the target and the defender should be retreated during the next turn.

Notice that although loops are avoided, there could still exist paths of attacks where a unit is attacking another unit that is moving towards another region. See an example in Figure 4.9. In order to let moving units to try to move before being attacked, we must start resolving the moves towards unoccupied targets and then proceed iteratively with the attacks towards those occupied targets that had already been resolved and so on until all movements are set to either feasible or unfeasible.


(a) Three unfeasible attacks against the green army that is feasibly moving. Its region remains standoff and the rest of units remain in their original position.

(b) Feasible attack of the red fleet that can move and dislodge the green army. The rest of moves are unfeasible and remain in their original position.

Figure 4.9: Examples of attacking paths.

When comparing the single strongest attacker strength with the defender strength we must take into account that there could be defenders that where ordered tomove but they are holding in the original position because they did not succeed performing the move. For those regions, the computed strength was intended to be used to attack not to defend. Thus, the strength of those defenders would be 1 independently of the number of units that where supporting its unfeasible move.

The *ResolveBattles* operation resolves ordinary battles iteratively using the *ResolveAttacks* operation to identify unresolved attacks and resolve them, and the *ApplyChanges* operation to update the *MovementsState* with new assessments, dislodgments and standoff provinces. To resolve battles, we start with those movements towards targets that:

1. are empty,
2. are not ordered to move, or
3. have their move order already assessed.

Then, we update the *MovementsState* and continue with those new movements that verify the previous constraints. And thus until no more movements satisfy them. This way we resolve moves before being attacked. As loops were already avoided, there is a finite number of iterations needed. Iterating this way, we can resolve attacks towards moving units knowing whether the attacked unit is indeed moving or not.

The set of movements that are not yet assessed is computed as follows: $\mathrm{dom}\, \mathit{assessedOrders} \ntriangleleft \mathit{move?}$

Therefore, we can declare the local variable M with the not yet resolved moves towards targets that are not performing not yet resolved moves:
$M == \mathrm{dom}\, \mathit{assessedOrders} \ntriangleleft \mathit{move?} \ntriangleright \mathrm{dom}(\mathrm{dom}\, \mathit{assessedOrders} \ntriangleleft \mathit{move?})$

Then, we proceed analysing every single move $ri \mapsto t \in M$ and resolving it as either feasible or unfeasible in the following way:

1. Movements towards targets that are not the single strongest are unfeasible. Those are detected with this line:
   $\neg(\forall\, rj : REGION \mid rj \neq ri \wedge rj \mapsto t \in move? \bullet strength(rj) < strength(ri))$

2. Single strongest moves towards targets are detected by:
   $\forall\, rj : REGION \mid rj \neq ri \wedge rj \mapsto t \in move? \bullet strength(rj) < strength(ri)$

   their resolution depend on whether the target is occupied or not.

   - Target empty or occupied with a unit performing a feasible move:
     $t \notin \operatorname{ran} control \vee t \in \operatorname{dom}(feasible \lhd move?)$

     Those single strongest moves towards targets that are empty or will be empty are feasible.

   - Target occupied with a unit that is not performing a feasible move:
     $t \in \operatorname{ran} control \wedge t \notin \operatorname{dom}(feasible \lhd move?)$

     The resolution of those movements depend on whether:

     – it is a countryman attack:
       $control^{\sim}(\!|\{ri\}|\!) = control^{\sim}(\!|\{t\}|\!)$
       A countryman attack cannot dislodge a unit. Countryman attacks are unfeasible.

     – it is not a countryman attack:
       $control^{\sim}(\!|\{ri\}|\!) \neq control^{\sim}(\!|\{t\}|\!)$
       Then we should compare the strength of the attack and the defence:

       ∗ if the attack is stronger than the defence:
         $strength(ri) > strength(t)$
         The move is feasible and the defender is dislodged.

       ∗ if the attack is less or equal strong than the defence.
         $strength(ri) \leq strength(t)$
         The move is unfeasible.

We formalise this complex operation using four simpler operations, one representing every single possibility. Thus, we define four operations with regions $ri?, t? : REGION$ as inputs and the boolean value $a! : Bool$ as output. Those input values must verify that:

- $ri?$ moves towards $t?$, that is: $ri? \mapsto t? \in move?$, and

- $t?$ is not performing a non yet assessed move, that is:

  $t? \notin \operatorname{dom}(\operatorname{dom} assessedOrders \ntriangleleft move?)$

The output stands for either feasible $a! = True$, or unfeasible $a! = False$. When introducing the operators below, we do not provide detailed explanation because that explanation was already given in the previous paragraphs.

Non single strongest moves are unfeasible. This is set by *ResolveNonSingleStrongest* operation.

ResolveNonSingleStrongest

$$\begin{array}{l}
\Xi \mathit{MovementsState} \\
ri? : \mathit{REGION} \\
t? : \mathit{REGION} \\
a! : \mathit{Bool} \\
\hline
ri? \mapsto t? \in \mathit{move}? \\
t? \notin \mathrm{dom}(\mathrm{dom}\, \mathit{assessedOrders} \lhd \mathit{move}?) \\
\neg\, (\forall\, rj : \mathit{REGION} \mid rj \neq ri? \land rj \mapsto t? \in \mathit{move}? \bullet \mathit{strength}(rj) < \mathit{strength}(ri?)) \\
a! = \mathit{False}
\end{array}$$

Single strongest moves towards empty targets or targets occupied by units that are feasibly moving are feasible as defined in *ResolveMove* operation.

ResolveMove

$$\begin{array}{l}
\Xi \mathit{MovementsState} \\
ri? : \mathit{REGION} \\
t? : \mathit{REGION} \\
a! : \mathit{Bool} \\
\hline
ri? \mapsto t? \in \mathit{move}? \\
t? \notin \mathrm{dom}(\mathrm{dom}\, \mathit{assessedOrders} \lhd \mathit{move}?) \\
\forall\, rj : \mathit{REGION} \mid rj \neq ri? \land rj \mapsto t? \in \mathit{move}? \bullet \mathit{strength}(rj) < \mathit{strength}(ri?) \\
t? \notin \mathrm{ran}\, \mathit{control} \lor t? \in \mathrm{dom}(\mathit{feasible} \lhd \mathit{move}?) \\
a! = \mathit{True}
\end{array}$$

When the target of a single strongest move is occupied by a unit that is not feasibly moving but that is controlled by the same power, the attack is a countryman attack and thus it is unfeasible. The *ResolveCountrymanAttack* operation deals with those countryman attacks.

ResolveCountrymanAttack

$\Xi MovementsState$

$ri? : REGION$

$t? : REGION$

$a! : Bool$

$ri? \mapsto t? \in move?$

$t? \notin \operatorname{dom}(\operatorname{dom} assessedOrders \ntriangleleft move?)$

$\forall rj : REGION \mid rj \neq ri? \land rj \mapsto t? \in move? \bullet strength(rj) < strength(ri?)$

$t? \in \operatorname{ran} control \land t? \notin \operatorname{dom}(feasible \lhd move?)$

$control^{\sim} (\!| \{ri?\} |\!) = control^{\sim} (\!| \{t?\} |\!)$

$a! = False$

Contrarily, when the single strongest attack is not a countryman attack, it is resolved comparing the attacking and defensive strength. As defined in the *ResolveNonCountrymanAttack* operation, the attack is feasible if the attacker is stronger than the defence. Otherwise, the attack is unfeasible.

ResolveNonCountrymanAttack

$\Xi MovementsState$

$ri? : REGION$

$t? : REGION$

$a! : Bool$

$ri? \mapsto t? \in move?$

$t? \notin \operatorname{dom}(\operatorname{dom} assessedOrders \ntriangleleft move?)$

$\forall rj : REGION \mid rj \neq ri? \land rj \mapsto t? \in move? \bullet strength(rj) < strength(ri?)$

$t? \in \operatorname{ran} control \land t? \notin \operatorname{dom}(feasible \lhd move?)$

$control^{\sim} (\!| \{ri?\} |\!) \neq control^{\sim} (\!| \{t?\} |\!)$

$strength(ri?) > strength(t?) \Rightarrow a! = True$

$strength(ri?) \leq strength(t?) \Rightarrow a! = False$

Finally, we combine those four operations to form the *ResolveAttack* operation defined as:

$ResolveAttack \mathrel{\widehat=} ResolveNonSingleStrongest \lor ResolveMove \lor ResolveCountrymanAttack \lor ResolveNonCountrymanAttack$

$ResolveAttack : REGION \times REGION \rightarrow\!\!\!\!\!\!+ Bool$

As this operation must be executed for any single movement $r \mapsto t \in M$, we define the *ResolveAttacks* operation that does it and outputs the currently being proceed movements $M!$ and their assessments as the partial function $A!$.

ResolveAttacks

$\Xi MovementsState$

$M! : REGION \mathrel{+\!\!\!\!\!\to} REGION$

$A! : REGION \mathrel{+\!\!\!\!\!\to} Bool$

$\textbf{let } M == \mathrm{dom}\, assessedOrders \unlhd move? \unrhd \mathrm{dom}(\mathrm{dom}\, assessedOrders \unlhd move?) \bullet$
$\quad \# M > 0 \land$
$\quad M! = M \land$
$\quad A! = \{r, t : REGION \mid r \mapsto t \in M \bullet r \mapsto ResolveAttack(r, t)\}$

Once the new assessments are computed, we can update the *MovementsState* aggregating the content of A to *assessedOrders* and computing the dislodgments and standoff provinces. The standoff provinces are those with regions that are targets remaining empty. That is, a target that:

- is empty or has a unit performing a feasible movement,
  $t \notin \mathrm{ran}\, control \lor t \in \mathrm{dom}(\mathrm{dom}(A \rhd \{True\}) \lhd move?)$

  and,
- there is no feasible movement towards it.
  $t \notin \mathrm{ran}(\mathrm{dom}(A \rhd \{True\}) \lhd move?)$

Dislodgments are computed as those movements with a unit in the target that is not performing a feasible move but that receive a feasible attack:
$dislodged' = \{t, r : REGION \mid t \in T \land t \in \mathrm{dom}\, control \land t \notin \mathrm{dom}(\mathrm{dom}(A \rhd \{True\}) \lhd move?) \land r \mapsto t \in (\mathrm{dom}(A \rhd \{True\}) \lhd move?) \bullet r \mapsto t\})$

The operation that updates the *MovementsState* is the *ApplyChanges* operation that has as inputs the currently being proceed movements $M?$ and their assessments as the partial function $A?$.

ApplyChanges

$\Delta MovementsState$

$M? : REGION \mathrel{+\!\!\!\!\!\to} REGION$

$A? : REGION \mathrel{+\!\!\!\!\!\to} Bool$

$dislodged' = dislodged \oplus \{r, t : REGION \mid$
$\quad t \in \mathrm{ran}\, control \land$
$\quad r \mapsto t \in \mathrm{dom}(A? \rhd \{True\}) \lhd M? \land$
$\quad t \notin \mathrm{dom}(feasible \lhd move?) \bullet r \mapsto t\}$

$standoff' = standoff \cup \{t : REGION \mid$
$\quad t \in \mathrm{ran}\, M? \land$
$\quad (t \notin \mathrm{ran}\, control \lor t \in \mathrm{dom}(feasible \lhd move?)) \land$
$\quad (\forall r : REGION \mid r \mapsto t \in M? \bullet A?(r) = False) \bullet province(t)\}$

$assessedOrders' = assessedOrders \oplus A?$

As the output variables if *ResolveAttacks* matches the input variables of *ApplyChanges*, we can combine those operations using a pipe and obtain the *ResolveBattles* operation:

$ResolveBattles \mathrel{\widehat{=}} ResolveAttacks >> ApplyChanges$

4.4.4 Updating the Game State

When the steps described in previous sections are done, all controlled regions have been assessed and the standoff provinces and dislodged units are computed. The last operation to perform in order to resolve movements is to update the game state. This is done by the *GameStateUpdate* operation using the feasibility and the dislodgment information to update the controlled regions. The current season changes either from Spring to Summer or from Fall to Autumn. Owned supply centres remain intact and the retreatement function is populated given the dislodged units and the standoff provinces.

Feasible moves advance letting the power control the destination of the move instead of the source. Unfeasible moves do not advance. If they are not dislodged, those units remains in the original positions that are the sources of the moves. The power will still control the unit in the same position. Other units that are not dislodged remain in the same region allowing the power to keep controlling the unit in the same position. Retreatements are calculated for each dislodged unit. Dislodged units will be able to retreat to any adjacent region that is not belonging to a province that:

- is a standoff,
- contains a controlled region, or
- contains the region source of the attack that dislodged the unit.

GameStateUpdate

$\Delta MovementsState$

$$\textbf{let } M == \operatorname{dom} assessedOrders \lhd move? \rhd \operatorname{dom}(\operatorname{dom} assessedOrders \lhd move?) \bullet \\ \#M = 0$$

$$control' = \\ \{p : POWER;\ r : REGION \mid p \mapsto r \in control \land r \in \operatorname{dom} move? \land \\ r \in feasible \bullet p \mapsto move?(r)\} \cup \\ \{p : POWER;\ r : REGION \mid p \mapsto r \in control \land r \in \operatorname{dom} move? \land \\ r \in unfeasible \land r \notin \operatorname{ran} dislodged \bullet p \mapsto r\} \cup \\ \{p : POWER;\ r : REGION \mid p \mapsto r \in control \land \\ r \notin \operatorname{dom} move? \land r \notin \operatorname{ran} dislodged \bullet p \mapsto r\}$$

$$own' = own$$

$$(currentSeason = Spring) \Rightarrow (currentSeason' = Summer)$$

$$(currentSeason = Fall) \Rightarrow (currentSeason' = Autumn)$$

$$retreatement' = \{ri, rj, rk : REGION;\ p : POWER \mid \\ ri \mapsto rj \in dislodged \land rk \neq ri \land rj \mapsto rk \in adj \land \\ province(rk) \notin standoff \land p \mapsto rj \in control \bullet (p, rj) \mapsto rk\}$$

Finally, the *ResolveMovements* operation combines all the operations that have to be done in a movement season to resolve the orders:
$ResolveMovements \mathrel{\widehat{=}} (InitOrders \land \Delta MovementsState) \mathbin{⨾} ComputeOrderStrength \mathbin{⨾} AvoidLoops \mathbin{⨾} ResolveBattles * \mathbin{⨾} GameStateUpdate$

4.5 Retreatements

The retreatement seasons are Summer and Autumn. These seasons are used by powers to retreat those units that have been dislodged in the previous movement season. In fact, all dislodged units must be ordered. The types of orders available at these seasons are: (1) *retreat* that is to move the unit to one of the available destinations, and (2) *disband* that is to remove the unit. The dislodged units can only be retreated to an adjacent empty region that was not standoff in previous movement season neither the original position of the dislodging unit. Alternatively, they can be ordered to disband. Thus disband is only mandatory when retreat is not possible.

The regions where a dislodged unit can retreat to are computed at the end of the movement season resolution and are included in the *GameState*. The resolution of the retreatement season consists in updating the controlled regions according to the retreat or disband input orders. There is only one kind of possible conflict that is when two units are ordered to retreat to the same region. In that case, they are both disbanded.

The *ResolveRetreatements* operation checks wether the current season is Summer or Autumn. It also verifies that all dislodged units are being ordered either to retreat or disband:

$$\operatorname{dom} \mathit{retreateTo}? \cup \mathit{disband}? = \{p : \mathit{POWER}; r : \mathit{REGION} \mid \\ (p, r) \in \operatorname{dom} \mathit{retreatement} \bullet r\}$$

And no one is ordered to do both:

$$\mathsf{disjoint}\ \langle \operatorname{dom} \mathit{retreateTo}?, \mathit{disband}? \rangle$$

Then it updates the *control* relation adding the units that have been succesfully retreated. A retreat succed when no other unit is retreating to the same position, that is:

$$\# \operatorname{dom}(\mathit{retreateTo}? \rhd \{rj\}) = 1$$

Failed retreats are disbanded, thus those units are not controlled any more. They are removed from the board.

Finally the current season is updated to Fall or Winter depending on wether the current season was Summer o Autumn respectively. The owned supply centers and the year do not change, and the retreatement info is reset.

$$
\begin{array}{|l}
\textit{ResolveRetreatements} \\
\hline
\Delta \textit{GameState} \\
\textit{retreateTo}? : \textit{REGION} \nrightarrow \textit{REGION} \\
\textit{disband}? : \mathbb{P}\, \textit{REGION} \\
\hline
\textit{currentSeason} = \textit{Summer} \lor \textit{currentSeason} = \textit{Autumn} \\
\operatorname{dom} \textit{retreateTo}? \cup \textit{disband}? = \{p : \textit{POWER};\ r : \textit{REGION} \mid \\
\quad (p, r) \in \operatorname{dom} \textit{retreatement} \bullet r\} \\
\mathsf{disjoint}\ \langle \operatorname{dom} \textit{retreateTo}?, \textit{disband}? \rangle \\
\textit{control}' = \textit{control} \oplus \{p : \textit{POWER};\ ri, rj : \textit{REGION} \mid \\
\quad ri \mapsto rj \in \textit{retreateTo}? \land \#(\operatorname{dom}(\textit{retreateTo}? \rhd \{rj\})) = 1 \land \\
\quad (p, ri) \in \operatorname{dom} \textit{retreatement} \bullet p \mapsto rj\} \\
(\textit{currentSeason} = \textit{Summer}) \Rightarrow (\textit{currentSeason}' = \textit{Fall}) \\
(\textit{currentSeason} = \textit{Autumn}) \Rightarrow (\textit{currentSeason}' = \textit{Winter}) \\
\textit{own}' = \textit{own} \\
\textit{currentYear}' = \textit{currentYear} \\
\textit{retreatement}' = \{\} \\
\hline
\end{array}
$$

4.6 Adjustments

There is only one adjustment season that is Winter. It is the last season of a year. In Winter, the ownership of supply centres is updated giving the ownership of every controlled supply centres to the power that is currently controlling it. By controlling a supply centre we mean that the power controls the unit that is hosted in one of the regions of a supply centre province.

The *UpdateOwnership* operation checks whether the current season is Winter. It declares the local variable O that relates every power with the provinces that it controls, $\textit{province} \circ \textit{control}$, that are supply centres, $sc(s) = pr$. The resulting relation is:

$$
O == \{p : \textit{POWER};\ pr : \textit{PROVINCE};\ s : \textit{SUPPLY\_CENTRE} \mid \\
\quad p \mapsto pr \in \textit{province} \circ \textit{control} \land sc(s) = pr \bullet p \mapsto s\}
$$

Then, it updates the own relation changing the ownership of the controlled supply centres to the controller:

$$
\textit{own}' = (\textit{own} \mathbin{\rhd\!\!\!-} \operatorname{ran} O) \cup O
$$

The control relation, the current season, and the current year do not change. Neither the retreatement info.

$UpdateOwnership$

$\Delta GameState$

$currentSeason = Winter$

$\mathbf{let}\ O == \{p : POWER; pr : PROVINCE; s : SUPPLY\_CENTRE \mid p \mapsto pr \in province \circ control \land sc(s) = pr \bullet p \mapsto s\} \bullet own' = (own \rhd \mathrm{ran}\, O) \cup O$

$control' = control$

$currentSeason' = currentSeason$

$currentYear' = currentYear$

$retreatement' = retreatement$

As a consequence of updating the supply centres ownership, the number of owned supply centres for some powers increase and for some others decrease. If there is a power owning half or more of the supply centres, that power wins and the game is over. The *WinnerFound* operation checks whether there is a winner and outputs it.

$WinnerFound$

$\Xi GameState$

$winner! : POWER$

$currentSeason = Winter$

$\exists p : POWER \mid \#(own(\!|\{p\}|\!)) \geq (numberOfSC \operatorname{div} 2) \bullet winner! = p$

If there is no winner yet, the game continues letting the powers adjust their number of supply centres with the number of units that they control. Those powers with more supply centres than units should be able to build more units until there will be as many units as owned supply centres. Those with more units than supply centres must remove as many units as necessary until there will be as many units as owned supply centres. Therefore, the orders available are: (1) *build* that is to create a new unit; (2) *remove* that is to remove a unit, and (3) *waive* that means not to create any more units.

Units must be created in empty owned homelands. That is, for a power to be able to create a unit in a given province:

- it has to be part of its homeland,
- it has to be owned by itself, and
- it cannot be hosting any other unit.

If that is the case, the power can create the unit in whatever region belonging to that province, or order a waive meaning that it is not going to create all units. It is often the case of powers that have more supply centres than units but that cannot build units because of missing empty owned homelands. In those cases, the power must order a waive order if it has the right to build units but not enough space for all them to be built.

The operation *ApplyAdjustments* receives as input the orders that the powers made public. Build orders are represented by the *build?* relation that assigns the region hosting the new unit to the power that is building it. Removes are represented by the set of regions *remove?* that contains the regions hosting the units that are being removed. And waive orders are represented by the set of powers *waive?* performing waives.

The operation starts checking whether the current season is Winter and whether the game is not over. If all powers have less than half of the supply centres owned, then the game is not over:
$\forall p : POWER \bullet \#(own(\!|\{p\}|\!)) < (numberOfSC \ \mathsf{div}\ 2)$

If that is the case, it continues verifying the input orders that must satisfy the following constraints:

- A power p can only build a unit in a region r belonging to a supply centre that is one of its homelands, $sc^{\sim}(province(r)) \in homeland(\!|\{p\}|\!)$, and that is currently owned by the same power, $sc^{\sim}(province(r)) \in own(\!|\{p\}|\!)$, and not controlled, $province(r) \notin \operatorname{ran} province \circ control$. That is, every build order $p \mapsto r \in build?$ must satisfy the following:

  $sc^{\sim}(province(r)) \in homeland(\!|\{p\}|\!) \cap own(\!|\{p\}|\!) \wedge$
  $province(r) \notin \operatorname{ran} province \circ control$

- A power p can build as many units as necessary to make the number of controlled units to be equal to the number of owned supply centres:

  $\#(build?(\!|\{p\}|\!)) = (\#(own(\!|\{p\}|\!)) - \#(control(\!|\{p\}|\!)))$

  Otherwise, it can build less than those necessary units and perform a waive:

  $\#(build?(\!|\{p\}|\!)) < (\#(own(\!|\{p\}|\!)) - \#(control(\!|\{p\}|\!)) \wedge p \in waive?)$

  We compact this constrain and say that for every power p that has less controlled regions than owned supply centres, $(control(\!|\{p\}|\!)) < \#(own(\!|\{p\}|\!))$:

  $\#(build?(\!|\{p\}|\!)) = (\#(own(\!|\{p\}|\!)) - \#(control(\!|\{p\}|\!))) \vee$
  $\#(build?(\!|\{p\}|\!)) < (\#(own(\!|\{p\}|\!)) - \#(control(\!|\{p\}|\!)) \wedge p \in waive?)$

- A power p must remove as many units as necessary to make the number of controlled units to be equal to the number of owned supply centres. This is verified requiring for every power with less owned supply centres than units, $\#(own(\!|\{p\}|\!)) < \#(control(\!|\{p\}|\!))$ to satisfy that the controlled regions removed must be the same needed to make that equal:

  $\#(control(\!|\{p\}|\!) \cap remove?) = \#(control(\!|\{p\}|\!)) - \#(own(\!|\{p\}|\!))$

Finally, the operation updates the control relation removing the units being ordered to remove and adding the new built units. Then, it also updates the current season to Spring and increment the year.

$$
\begin{array}{|l}
\textit{ApplyAdjustments} \\
\hline
\Delta \textit{GameState} \\
\textit{build}? : \textit{POWER} \leftrightarrow \textit{REGION} \\
\textit{remove}? : \mathbb{F}\ \textit{REGION} \\
\textit{waive}? : \mathbb{F}\ \textit{POWER} \\
\hline
\textit{currentSeason} = \textit{Winter} \\
\forall p : \textit{POWER} \bullet \#(\textit{own}(\!| \{p\} |\!)) < (\textit{numberOfSC}\ \mathsf{div}\ 2) \\
\textit{remove}? \subseteq \operatorname{ran} \textit{control} \\
\forall p : \textit{POWER};\ r : \textit{REGION} \mid p \mapsto r \in \textit{build}? \bullet \\
\quad \textit{sc}^{\sim}(\textit{province}(r)) \in \textit{homeland}(\!| \{p\} |\!) \cap \textit{own}(\!| \{p\} |\!) \wedge \\
\quad \textit{province}(r) \notin \operatorname{ran}(\textit{province} \circ \textit{control}) \\
\forall p : \textit{POWER} \mid \#(\textit{control}(\!| \{p\} |\!)) < \#(\textit{own}(\!| \{p\} |\!)) \bullet \\
\quad \#(\textit{build}?(\!| \{p\} |\!)) = \#(\textit{own}(\!| \{p\} |\!)) - \#(\textit{control}(\!| \{p\} |\!)) \vee \\
\quad (\#(\textit{build}?(\!| \{p\} |\!)) < \#(\textit{own}(\!| \{p\} |\!)) - \#(\textit{control}(\!| \{p\} |\!)) \wedge p \in \textit{waive}?) \\
\forall p : \textit{POWER} \mid \#(\textit{own}(\!| \{p\} |\!)) < \#(\textit{control}(\!| \{p\} |\!)) \bullet \\
\quad \#(\textit{control}(\!| \{p\} |\!) \cap \textit{remove}?) = \#(\textit{control}(\!| \{p\} |\!)) - \#(\textit{own}(\!| \{p\} |\!)) \\
\textit{control}' = (\textit{control} \rhd \textit{remove}?) \cup \textit{build}? \\
\textit{currentSeason}' = \textit{Spring} \\
\textit{currentYear}' = \textit{currentYear} + 1
\end{array}
$$

The following *ResolveAdjustments* operation combines the three previous operations as follows:
$\textit{ResolveAdjustments} \mathrel{\widehat=} \textit{UpdateOwnership} \mathbin{;} \textit{WinnerFound} \vee \textit{ApplyAdjustments}$

Once all the operations for resolving every single part of the adjudication are defined, we can combine them to form the *AdjudicatorAlgorithm* operation:
$\textit{AdjudicatorAlgorithm} \mathrel{\widehat=}$
$(\textit{Init} \wedge \Delta \textit{GameState}) \mathbin{;} (\textit{ResolveMovements} \mathbin{;} \textit{ResolveRetreatements} \mathbin{;}$
$\textit{ResolveMovements} \mathbin{;} \textit{ResolveRetreatements} \mathbin{;} \textit{ResolveAdjustments})*$

4.7 Complexity

To analyse the complexity of The Diplomacy Game we focus on the movement seasons of the standard version of the game without convoys. We compute the complexity of the game as the branching factor of its extensive form representation. Even omitting the negotiation, we can see that the branching factor of The Diplomacy Game compared to the branching factor of Chess and Go is enormous.

In The Diplomacy Game, all players perform their movements at the same time. Any player has several units and has to select an order for each of them. The game starts with 7 players and a total number of 22 units. As turns go by, some players lose all their units and thus are eliminated. Others enlarge their power controlling more and more units. The number of units on the board is limited by the number of supply centres as it is necessary to own one supply centre in order to control a unit. The standard map contains 34 supply centres, thus 34 is the maximum number of units that can be over the map. As there are rules to be satisfied in order to build a unit that is having own but empty homeland provinces available, we computed the average number of units per turn in a large bunch of games and obtained a number of units per turn in average equal to 30. Therefore, an order has to be assigned to each of the 30 units that are on the board in average per turn.

The movements that a unit can perform include a hold, several moves, some supports to hold and supports to move. The number of moves that a unit can perform depend on the number of adjacent regions that its hosting region has. We use the terms *hosting region* and *hosting province* to refer to the region and the province where a given unit is located. We computed the average number of adjacency regions that a region in the standard Diplomacy map has and it is a number near to 4. Remember that not all provinces that are adjacent to the hosting province have a region that is adjacent to the hosting region. This is so because, for example, an army in a coast cannot move towards the sea. Curiously, some provinces that are adjacent to the hosting province contain more than one region that is adjacent to the hosting region. This is the case when a unit can move to both of the coastal regions of a bi-coastal province. As there are only 3 bi-coastal provinces compared to the total number of 75 provinces, we ignore the case of bi-coastal provinces in the subsequent calculations. In conclusion, we assume that the number of regions adjacent to a particular region is 4 and thus, the number of moves that a unit can perform is in average 4.

For a unit to be able to support other units holding, it must have neighbours that are units situated in regions belonging to provinces that contain a region adjacent to the hosting region. We analysed a large number of games and calculated that in average the number of neighbours is 2. Therefore, the number of supports to hold that a unit can perform is 2. The case of supports to move is more complicated to compute as a unit can support to move any unit moving towards every region belonging to the provinces with adjacent regions to the hosting region. We use a simplified abstract version of the map only with regions to illustrate the problem and show how we computed the number of possible supports to move, see Figure 4.10. Notice that in this map every region has 4 adjacent regions and every unit has 2 neighbours. `x can support units `y and `z to move towards itself or its adjacent regions.[5] Therefore, every unit `y can be supported to move towards 3 different regions, and every unit `z can only be supported to move towards 1 region. In overall we have that `x can support $2 \times 3 + 2 = 8$ moves.

Counting together all possible movements of a unit we have 1 hold, 4 moves, 2 supports to hold and 8 supports to move that sum $1 + 4 + 2 + 8 = 15$ possible movements. Having 30 units per turn to be ordered and an average of 15 possible orders per unit means that the branching factor is $15^{30} = 1.43 \cdot 10^{22}$ that is just huge.[6] Remember that the branching factor for Chess is around 35 and Go is around 200.

[5]Supporting a unit to move towards the supporting unit is a possible order although it is not feasible. It is possible to order such a support but it has no effect as the rules of the game say that this is a countryman attack support and those supports must be cancelled during the resolution of orders performed with by the adjudicator algorithm.

[6][Kemmerling et al., 2011] provides a different complexity analysis with similar branching factor results although we differ computing the number of available orders per unit.

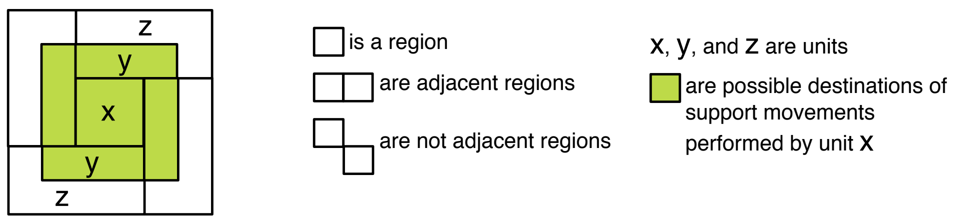


Figure 4.10: Abstract representation of the regions in the Diplomacy map. Unit `x is the given unit, units named `y are its neighbours and units `z are the units that can receive supports to move from `x .

Fortunately, not all possible orders are equiprobable. Some of them are more likely to be used than others and it is possible to analyse the behaviour of the other players during the game and negotiate with them in order to obtain more information that help us to guess what the other players are going to order to their units. The Diplomacy Game is not a game to be solved using computational power. This game would be solved when a program, a software agent, would be able to understand human relationships, their reaction and would be able to take advantage of this information in order to proficiency negotiate with them. Negotiating is just crucial to cope with the huge space of solutions and the uncertainty on the other players next orders.

4.8 Diplomacy as an RNP

Diplomacy is an example of Resource Negotiation Problem (RNP). It involves several players that repeatedly negotiate. All players perform their actions at the same time. And the actions are made public so anyone can check whether its negotiating counterparts honour the commitments reached at negotiation time.

To represent Diplomacy as an RNP we just need to use the players (powers) as agents, and the units as the resources that they control. The operators that can be applied to resources are the different order types, and unit orders are the actions. This is specially obvious during a movement season when the agents concurrently negotiate to reach agreements over the movements that they may jointly perform (e.g. getting the support of another player s unit to strengthen one of our units that will most probably get into battle in exchange of moving another of our units away from a certain province).

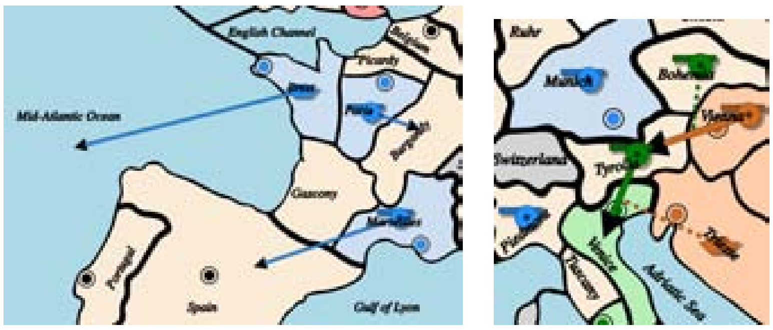
The negotiation that takes place during the movement seasons is usually performed in a natural language, for instance in English, as it is a game that is normally played by humans. Now, as an RNP, the negotiation has to be limited to plans of action. To illustrate the notion of plan we use a simple ontology illustrated in Figure 4.11 that allows to represent orders.

In Figure 4.12 we graphically represent two plans, on 4.12a an individual plan for France (in blue) and on 4.12b a joint plan between Italy (in green) and Austria (in brown).

The plan for France is to move from Paris to Burgandy, from Brest to Mid-Atlantic Ocean and from Marseilles to Spain. Thus, the plan $p = \{a_1, a_2, a_3\}$ is

year ::= $\underline{\text{integer}}$
season ::= spr | sum | fal | aut | win
power ::= fra | eng | tur | rus | ita | aus | ger
coast ::= ncs | scs | ecs | wcs
regionType ::= amy | sea | coast
supplyCenter ::= spa | mar | par | stp | ...
province ::= *supplyCenter* | gas | bur | sil | tus | ...
region ::= Region(*province*, *regionType*)
unit ::= Unit(*power*, *region*)
order ::= hld(*unit*) | mto(*unit*, *region*) | sup(*unit*, hld(*unit*)) | sup(*unit*, mto(*unit*, *region*)) | rto(*unit*, *region*) | dsb(*unit*) | bld(*unit*) | rem(*unit*) | wve(*power*)
offer ::= pce($power^+$) | aly($power^+$, $power^+$)

Figure 4.11: Diplomacy ontology.



(a) Plan (b) Joint plan

Figure 4.12: Two examples of plans.

a set of movements.[7]

$$a_1 = \texttt{mto(Unit(fra, Region(par, amy)), Region(bur, amy))}$$
$$a_2 = \texttt{mto(Unit(fra, Region(bre, flt)), Region(mao, sea))}$$
$$a_3 = \texttt{mto(Unit(fra, Region(mar, amy)), Region(spa, amy))}$$

In Diplomacy we cannot move the same unit to two different regions at the same time. This is an example of non compatibility actions. Thus, given the action a_4 below, comp(a_1, a_4) does not hold and therefore neither feasible(p) holds if p contains a_1 and a_4 nor comp$(\{a_1\}, \{a_4\})$ holds.

$$a_4 = \texttt{mto(Unit(fra, Region(par, amy)), Region(gas, amy))}$$

Assuming that Austria controls only two units, Figure 4.12b illustrates an example of joint plan for Austria $p = \{a_5, a_6, a_7, a_8\}$ where actions are represented as:

$$\begin{aligned} a_5 &= \texttt{mto(Unit(ita, Region(tyr, amy)), Region(ven, amy))} \\ a_6 &= \texttt{sup(Unit(aus, Region(tri, flt)),} \\ &\qquad \texttt{mto(Unit(ita, Region(tyr, amy)), Region(ven, amy)))} \\ a_7 &= \texttt{mto(Unit(aus, Region(vie, amy)), Region(tyr, amy))} \\ a_8 &= \texttt{sup(Unit(ita, Region(mar, amy)),} \\ &\qquad \texttt{mto(Unit(aus, Region(vie, amy)),Region(tyr, amy)))} \end{aligned}$$

[7]To simplify notation we represent the names of powers and provinces in a compressed way. For instance, France is “fra”, Paris is “par” and Mid-Atlantic Ocean is “mao”.

Given this plan, Italy could propose Austria different options:

- propose(`ita`, `aus`, [Commit(`aus`,`ita`, Do(a_6))]). Italy proposes Austria a deal where Austria commits to support the Italian move from Tyrol to Venice. As Italy does not give anything in exchange it may not be effective.
- propose(`ita`, `aus`, [Commit(`ita`,`aus`, Do(a_5)), Commit(`aus`,`ita`, Do(a_6))]). This looks fairer, but Italy is actually not providing anything beneficial to Austria.
- propose(`ita`, `aus`, [Commit(`ita`,`aus`, Do(a_5)), Commit(`aus`,`ita`, Do(a_6)), Commit(`aus`,`ita`, Do(a_7)), Commit(`ita`,`aus`, Do(a_8))]). This proposal is more fair as Italy is also helping Austria.

Which one of these options should Italy actually send to Austria is what the negotiation strategy determine.

4.9 Summary

The board of The Diplomacy Game represents a map of Europe with four kinds of provinces: sea, inland, coast and bicoastal provinces. There are seven players that are denoted by powers. Each power has several chips representing military units that can be armies or fleets. Those units are spread along the map having no more than a unit per province. Armies cannot be located in sea provinces and neither fleets in inland provinces. We specify the map using a new concept, a region, that represents a potential location of a type of units in a province. The adjacency relation between neighbouring provinces is exported to regions corresponding to the same type of units in adjacent provinces. The corresponding region adjacency map describes the allowed unit moves between regions, i.e. between provinces. Some non sea provinces are special and denoted supply centres. A subset of the supply centres is partitioned between the powers and correspond to the homeland of those powers. Supply centres can be owned by the powers. All the previous information and the starting units describing the starting position of the game represent the static part of the game.

The dynamics of the game stablishes a sequence of turns divided in years and seasons that can be of three different types: movement, retreatement and adjustment. The game starts with the starting units placed on the homelands of the powers. Then, all powers decide what moves to perform and, next, the game proceeds resolving possible conflicts and generating the next turn. The information of the game corresponding to a particular turn is denoted by the game state. It contains the current year and season, the location of units on the map, the owned supply centres per power, and information about previous retreatments. The moves that powers can order to their units are denoted by *orders*. Several types of orders are allowed per season type. We specify the adjudicator algorithm responsible of the resolution of conflicts and the generation of the next game state from the set of orders selected by the powers in parts, a part per season type.

The orders allowed in movement seasons are to hold, to move, to support another unit holding and to support another unit moving to a given destination. The resolution of this season use to generate multiple conflicts and requires special attention. Several kinds of battles can be detected and, although their resolution is deterministic, it requires of several steps including the computation of unit strengths. The strength of a unit is computed as the number of supports that the unit receives in each moment. After a movement season, a retreatement season takes place. Units dislodged in the previous movement season are included in the retreatements. Those must be retreated or disbanded in this season. Next, either another movement season or the adjustment season takes place. There are two movement and retreatement seasons per year, but only an adjustment season. The adjustments consist in updating the assignation of owned supply centres and creating or removing units according to the difference between owned supply centres and controlled units per power. After adjustments, those numbers should be equal. Even thought, other rules apply for building units that limit this equilibrium between supply centres and units per power. The game ends when a power has a number of supply centre equal or larger than half the number of supply centres in the map. Only one player can win the game.

The computational complexity of the game is huge. The branching factor is 15^{30} that is astronomical compared to Chess and Go (35 and 200 respectively). The Diplomacy Game requires a different kind of approximations that take into account the study of the other players movements and, mainly, their negotiations. Taking every single negotiation step into account, the branching factor would be overwhelming. Interestingly, the complex negotiations of Diplomacy are the key for success. A good understanding of the negotiations and the other players provides the formula for solving this game.

The Diplomacy Game can be seen as an RNP. If we focus on the movement season, we can see units as resources and orders as actions to be applied on those resources. Then, the negotiation is constrained to trading offers of sets of orders to perform jointly. For instance, we can request the help in performing a particular move, and even provide some interesting move for the other agent.

Chapter 5

HANA architecture

HANA is a software architecture for agents that need to negotiate joint plans of action in realistic scenarios such those in RNPs. These negotiations may involve humans and repeat along time. The chapter describes the agent architecture (Section 5.1), and its main modules that are: the world model (Section 5.2), the plan search (Section 5.3), and negotiation (Section 5.4). The chapter terminates with a summary (Section 5.5).

5.1 Agent architecture

In this section, we describe a software architecture to build agents capable of participating in RNPs. We introduce here its main modules and then we give details for each of them in subsequent sections. We refer to the architecture as *HANA* and to the agents designed according to HANA as *HANA agents*.[1] The architecture is graphically represented in Figure 5.1.

In Chapter 3, we define RNPs and explain that they happen in fully observable environments that agents can perceive and where agents can execute actions whose (deterministic) consequences are also observable. Furthermore, the environment allows agents to exchange private messages. Thus, the first component of HANA is an *interface module* that situates the agents in their environment, that is: it allows to observe the environment state, observe and execute actions, and exchange messages with other agents. In other words, this module contains the sensors and actuators of the agent. Which actions to execute and which messages to send is decided by the *negotiation module*.

The design philosophy behind HANA is to provide some means to negotiate as humans do, as the negotiation counterparts could be humans. In particular, there are two capabilities that we think realistic agents should show: dealing with emotions, and dealing with uncertainty [Minsky, 1999]. The architecture incorporates emotions as this is an important part of the non-constructivist rationality approach, we need to understand emotional reactions of the other negotiators. Although the environment is fully observable, the actions to be executed by the other agents can only be guessed analysing the other agents

[1]HANA is an acronym for Human-Aware Negotiation Architecture. We call the architecture this way to stress the aim of negotiating with humans in realistic scenarios that motivated the design of this agent architecture.

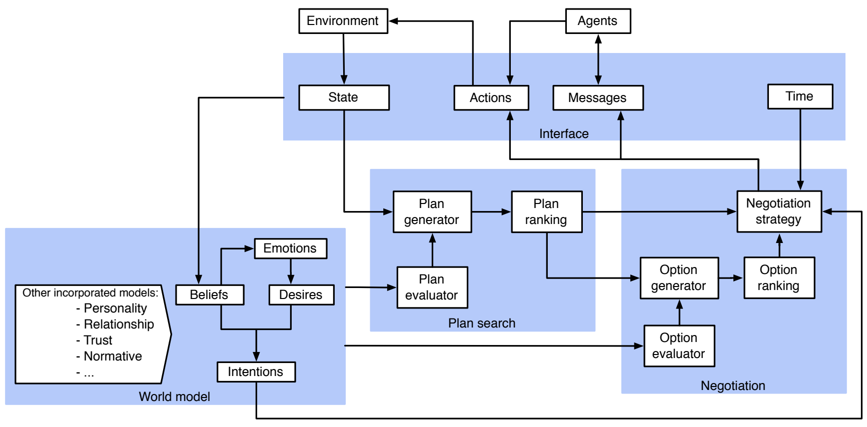


Figure 5.1: Graphical representation of HANA. Arrows represent data flows. Coloured boxes represent the modules that form part of the agent, and white boxes are components of those modules.

previous behaviour. To cope with this uncertainty, we decided to represent the world as a graded BDI model, that is with graded *beliefs*, *desires* and *intentions* following the g-BDI model [Casali et al., 2011]. In Section 5.2 we provide a more in-depth description of the *world model module* that is the one containing emotions and BDI components.

The space of plans and negotiation options that an agent can execute and propose, respectively, is potentially huge, as for example in The Diplomacy Game. Thus, we assume that the space is large enough and the negotiation time short enough to preclude obtaining the optimal. That means that any architecture for this type of negotiation needs to give the means to look for good enough solutions. Moreover, the longer it takes to decide what to propose, the less probable it is the proposal to be accepted. As time goes by, the agents reach agreements that increase the amount of commitments and reduce the set of options compatible with those commitments. Increasing acquired commitments increases, in turn, the probability that our desired plans will not be compatible any longer. Consequently, the architecture must allow to start negotiating from the very beginning of a negotiation round. Dealing with huge solution spaces is not an inconvenient for human agents, e.g. in playing Chess or Go. Humans do work with good enough solutions in their everyday lives. Time constraints, boredom, or tiredness make humans accept good enough solutions. To start negotiating from the very beginning, HANA proposes to perform a search&negotiation technique that assumes the plan search to go hand in hand with the negotiation. The *plan search module* executes an anytime algorithm that provides a periodically updated ranking of good enough plans. The ranking takes into account the commitments obtained by the *negotiation module*. And the negotiation module proposes options generated from the previously found good enough plans that contain actions to be executed by other agents. In this way, HANA agents can start the negotiation from the very beginning proposing options that, once negotiated, will provide new information because the option will be accepted or rejected to focus the search on the generation of new and better evaluated plans. As can be seen in Figure 5.1, the plan and option evaluators depend not only on the commitments but on the whole world module. Thus, those evaluation functions are also updated taking into account the intentions generated from new observations. The intentions trigger the decisions of the agent. In the following sections we provide a more in-depth description of the modules, their components and the decisions to be taken by the agent.

The execution of HANA consists of several concurrent processes for: the *interface* (to receive messages and observe the results of actions and the environment state), the *world model* (to update the world model given the perceived changes), the *plan search* (to continuously update the ranking of plans), and the *negotiation* (to generate options from plans and determine what to do next).

5.2 World model

For negotiation decisions to be effective, they need to be based on an accurate assessment of the state of the environment, and on the preferences of the other agents. Although the environment may be precisely known in certain domains, the preferences of others may be unknown, or may be uncertain. Moreover, specially in competitive scenarios, agents may lie about their preferences. This imposes the requirement that the world model has to be based on some uncertainty reasoning mechanism. During the last decade, many successful representation models have been based on BDI representations. Most work on BDI models has concentrated on providing agent-oriented programming frameworks and languages such as Jason [Bordini et al., 2007], Jack [Winikoff, 2005], AgentSpeak [Rao, 1996; d Inverno and Luck, 1998], 3APL [Hindriks et al., 2000] or 2APL [Dastani, 2008]; and on logical approaches to the BDI model such as modal logic [Rao and Georgeff, 1991], first-order logic [Rao, 1996], or belief degrees [Parsons and Giorgini, 1999; Pilotti et al., 2011]. BDI is based on the theory of human practical reasoning [Bratman, 1999] and has well-founded logical semantics [Rao and Georgeff, 1998]. The work of Casali et al. [Casali et al., 2011] on what they denoted by *g-BDI*, gives a powerful generic representation for degrees of belief, degrees of desire (preferences) and degrees of intention (active goals). We adapt this work to our problem and incorporate the beliefs, desires and intentions as the main components of the world model.

We consider that desires and intentions are derived from beliefs. What happens in the world determines whether we desire it to change into a different world, and whether we intend to make that happen. In this section, we concentrate on how HANA agents represent their beliefs about the next environment state. The most important aspect of the evolution of the world is what an agent expects to happen in the environment due to the decisions of other agents. This is so because the natural evolution of environments is subject to shared knowledge on physical laws, and thus known by every agent. Therefore, the evolution of the world can be due to either actions, A, or utterances, M, of other agents. We denote those events (actions and utterances) by $\Phi = M \cup A$. The agent has at all time a belief degree assigned to every element of Φ meaning how certain is the agent that that event will happen. We decided to model these degrees as probabilities because the data for them comes from the previous interactions with the other agents and thus those data can be statistically processed. Axiomatics on how to represent probabilities are provided in [Casali et al., 2011].

Definition 16 *Given $\Phi = M \cup A$, a* belief *is a tuple $\langle \alpha, \varphi, \vartheta \rangle \in \mathcal{A} \times \Phi \times [0,1]$ where ϑ is α's belief degree on φ happening. We denote by $\mathcal{B}$ the set of all possible beliefs and by $\mathcal{B}_\alpha \subseteq \mathcal{B}$ the set of possible beliefs of α.*[2]

We define the feasibility of $\varphi \in \Phi$ as an extension of the feasibility on utterances and actions.

Definition 17 *Given the current state $\omega \in W$ and the current dialogue Ψ, we define feasibility for $\varphi \in \Phi$ as:*

$$\text{feasible}(\varphi) = \begin{cases} \text{feasible}(\omega, \varphi) & \textit{if} \quad \varphi \in A \\ \text{feasible}(\Psi, \varphi) & \textit{if} \quad \varphi \in M \end{cases}$$

The particular type of problem studied here gives some restrictions on over what it is feasible to happen. All agents know the physical laws of the world, thus they all know that non feasible events will not happen. The degree of belief on a non feasible event would be then the minimum, 0. Also for plans: non feasible plans will not happen as those plans contain non feasible or non compatible actions. In the following we show two constraints for values of the belief model that are derived from the notion of feasibility and compatibility. In particular, only one operator can be applied to a resource. Two feasible actions operating on the same resource are non compatible and, thus, they cannot both happen at the same time. As we decided to represent degrees as probabilities, the summation of probabilities on all those actions to be operated on the same resource must be 1:

$$\forall \omega. \forall \langle \alpha, r \rangle \in \omega. \sum_{\{a_i = \langle \alpha, op_i, r \rangle | \text{feasible}(a_i, \omega) \text{ and } op_i \in Op\}} B_\alpha(a_i) = 1 \tag{5.1}$$

Moreover, the negotiation protocol proposed in this work imposes that only feasible utterances are possible.

$$\forall \Psi. \sum_{\{\mu_i | \text{feasible}(\Psi, \mu_i)\}} B_\alpha(\mu_i) = 1 \tag{5.2}$$

For a given environment state ω, the belief degree of α on an action a happening is $B_\alpha(a)$. Recall that plans are considered sets of actions that are to be executed at the same time. The belief on the execution of a feasible plan $p = \{a_1, a_2, \ldots, a_n\} \in P^\omega$ is thus naturally modelled according to HANA as the belief on the conjunction of the execution of each action that is then modelled as the product.

$$B_\alpha(p) = B_\alpha(a_1 \wedge a_2 \wedge \cdots \wedge a_n) = \Pi_{a_i \in p} B_\alpha(a_i) \tag{5.3}$$

When new observations of the environment are made, HANA agents update their beliefs. From the many possible *belief review functions* available in the literature [Alchourròn et al., 1985; Hansson, 2001; Pilotti et al., 2011] HANA uses a recency based belief revision defined as follows.

[2]We will note $\langle \alpha, \varphi, \vartheta \rangle \in \mathcal{B}$ as $B_\alpha(\varphi) = \vartheta$ when useful.

Definition 18 *Given an agent $\alpha \in \mathcal{A}$, a* belief review function*, denoted by $\sigma : 2^{\mathcal{B}_\alpha} \times 2^{\mathcal{B}_\alpha} \to 2^{\mathcal{B}_\alpha}$, is any function satisfying:*

- $\sigma(\mathcal{B}', \mathcal{B}'') = \mathcal{B}'''$
- *$\mathcal{B}'$ is the original belief set*
- *$\mathcal{B}''$ is a new belief set*
- *If $\langle \varphi, \vartheta \rangle \in \mathcal{B}'$, $\langle \varphi, \vartheta' \rangle \in \mathcal{B}''$ and $\vartheta \neq \vartheta'$ then $\langle \varphi, \vartheta' \rangle \in \mathcal{B}^*$*
- *If $\langle \varphi, \vartheta \rangle \in \mathcal{B}'$ and $\langle \varphi, \_ \rangle \notin \mathcal{B}''$ then $\langle \varphi, \vartheta \rangle \in \mathcal{B}^*$*
- *Nothing else belongs to $\mathcal{B}^*$*
- *$\mathcal{B}'''$ is the normalization[3] of $\mathcal{B}^*$*

Desires, intentions and emotions[4] are also updated via a similar simple update method that we omit here. Arrows in Figure 5.1 show the influence between the different motives: changes in the environment provoke updates in the belief set, that generates emotional updates, that update the desires. The new set of beliefs and desires determines new intentions.

The world model internal structure allows to represent complex behaviours as next example show.

Example 3 *The HANA agent* Revenger *plays Diplomacy and is configured to have a sensitive and aggressive personality. It comes to believe that the agent* Jaume *is an enemy as he executed a movement that attacks one of Revenger's units, and he never accepted any proposal in the past three negotiation rounds. Its personality rules for revenge trigger a desire (with a high degree) to damage Jaume, that will in turn give an intention (again with high degree) to attack one of Jaume's units although it is a very difficult task, and there are other alternative plans that would be easier to reach, and would give Revenger a higher rational utility.*

Although only a few components in Figure 5.1 are interconnected to build up the world model of HANA agents (beliefs, desires, intentions and emotions), other models might be incorporated. The world model is based on multicontext systems [Casali et al., 2011] that are modular structures that allow for an easy interconnection of different (logical) models, using transition functions between them, to build even more complex agents. For instance, a trust model may impact on intentions, as the intention degree to satisfy a desire via a plan with an untrustworthy agent should be low. Also, a social model might impact on intentions, as we might want to have a higher intention degree on plans involving

[3]To normalise, we follow the work done in [Fabregues et al., 2009], and compute the minimum relative entropy probability distributions with respect to the distributions in $\mathcal{B}'$ that consider the new beliefs in $\mathcal{B}''$ as constraints to satisfy, and that satisfy equations 5.1 and 5.2.

[4]To model emotions we follow [Revelle and Scherer, 2010] and represent *emotional states* or *moods*. The personality of the agent is part of the emotional model, and determines how to update the mood of the agent according to the beliefs about the environment. Personality may connect, for instance, frustration (a belief on a failed negotiation) with retaliation (a desire of a non-rational future negotiation).

an agent with whom we would like to increase the level of intimacy [Sierra and Debenham, 2007] than otherwise.

As defined in Section 3.2, the agents must fulfil a negotiation protocol in order to be able to negotiate with other agents. The rules or constraints that the protocol provides can be incorporated in the agent as internal norms to follow. This is done, according to HANA with a high desire degree on fulfilling the negotiation protocol, and some transition functions between this desire, and several beliefs that modify the degree of what we call *basic intentions*: *reply* δ (when the agent beliefs that it received proposal δ), *propose* (when there is time left in the negotiation round) and *executeActions* (when we are approaching the end of the negotiation round).

5.3 Plan search

The interplay between search and negotiation is the most important contribution of HANA. In most multi-issue negotiation problems the space of potential solutions is determined by the admissible values of issues. That is: potential solutions are elements of the cartesian product of the admissible values for each issue [Faratin et al., 1998b]. Differently, in RNPs the space of potential solutions is defined as the combination of feasible actions that are compatible. Only certain subsets of the space of actions constitute feasible solutions, and finding which ones are feasible is not straightforward. Good and bad solutions are not placed together nicely as in continuously valued attributes (e.g. if a certain low price is good, nearby low prices will be similarly good). Sometimes a small change in a plan makes it go from excellent to catastrophic. Moreover, the space of potential solutions in real domains is frequently huge. What HANA brings in to address this type of negotiation problem is a search&negotiation technique that enables the negotiation to start as soon as possible over reasonably good solutions. We explain the details of how solutions are sought in this section.

The outcome of the search process is a continuously refreshed ranking of candidate plans. The *plan ranking* is made by the *plan generator* thanks to a utility function that represents the preferences of the agent and that is implemented within a component of the architecture called *plan evaluator*, see Figure 5.1. Preferences are determined by the world model, and thus, they for instance take into consideration personality traits and relationships between agents. They do not evaluate only the individual position improvement.

Every agent α in a state transition system Ω must decide what actions to perform, that is, what complete plan $\bar{p}_\alpha$ to perform. Remember that given a state ω, next state ω' is computed by a state transition function $\mathbf{T} : W \times P \to W$. $\mathbf{T}(\omega, \bar{p})$ is defined for complete plans, $\bar{p} \in \bar{P}^\omega$, that are those that can be obtained from the union of complete plans for every agent controlling resources in the current state: $\bar{p} = \bigcup_{\beta \in \mathcal{A}} \bar{p}_\beta$. To decide what plan to perform, α must know what is the utility that every plan would provide. If α knew the plans of the other agents, $Q = \bigcup_{\beta \in \mathcal{A} \setminus \{\alpha\}} \bar{p}_\beta$, it could compute the utility of α performing the complete plan $\bar{p}_\alpha$ using its utility function, $\mathcal{U}_\alpha : W \to [0,1]$, as:

$$\mathcal{U}_\alpha(\omega, \bar{p}_\alpha) = \mathcal{U}_\alpha(\mathbf{T}(\omega, Q \cup \bar{p}_\alpha))$$

However, whilst agents may have a clear idea of their preferences, and hence can build up a utility function, it is usually impossible to know what other agents plans will be. Therefore, instead of using the deterministic transition function, $\mathbf{T} : W \times P \to W$, α has to use a probabilistic state transition function.

Definition 19 *Given a transition system $\Omega = \langle \mathcal{A}, R, Op, W, P, \mathbf{T}, \omega_0, W_f \rangle$, a* probabilistic transition function*, denoted by $\mathbb{T}(\omega' \mid \omega, p) \in \mathbb{P}(W)$ is any conditional probability distribution over W given $\omega \in W$ and $p \in P$, such that for every complete plan $\bar{p} \in \bar{P}^{\omega}$ then $\mathbb{T}(\mathbf{T}(\omega, \bar{p}) \mid \omega, \bar{p}) = 1$ and $\mathbb{T}(\omega' \mid \omega, \bar{p}) = 0$ for all $\omega' \neq \mathbf{T}(\omega, \bar{p})$.*

In RNPs, the state transition function $\mathbf{T} : W \times P \to W$ is fixed and common to all the agents. Instead, a probabilistic transition function has to be particular for each agent as it necessarily depends on the interpretation of other agents past behaviour. Therefore, we will denote by $\mathbb{T}_\alpha(\omega' \mid \omega, p)$ the probabilistic transition function of agent α. Then, the utility of a plan, complete or not, for an agent can be estimated as follows:

$$E[\mathcal{U}_\alpha(\omega, p)] = \sum_{\omega_i \in W} \mathbb{T}_\alpha(\omega_i \mid \omega, p) \times \mathcal{U}_\alpha(\omega_i) \tag{5.4}$$

The complexity here relies on the evaluation of $\mathbb{T}_\alpha(\omega' \mid \omega, p)$.

We can identify the problem of learning the probabilistic state transition function for all complete plans for a given agent $\alpha \in \mathcal{A}$ as a Markov Decision Process (MDP), [Kaelbling et al., 1998]. A MDP is a tuple $\langle S, A, L : A \times S \times S \to [0,1], R : A \times S \times S \to [0,1] \rangle$, where S is a finite set of states, A is a finite set of actions, $L(a, s, s')$ is the probability that action a in state s will lead to state s', and $R(a, s, s')$ is the immediate reward received after transition to state s' from state s with transition probability $L(a, s, s')$. The interpretation as an MDP is based on having $S = W$, $A = P$, $L = \mathbb{T}$ and $R(p, \omega, \omega') = \mathcal{U}_\alpha(\omega') - \mathcal{U}_\alpha(\omega)$. Learning $\mathbb{T}$ requires a wealth of data that is usually not available, and an initially random behaviour that may produce very negative outcomes in RNPs. Moreover, there is a required feature for any MDP problem that is not verified in our case: the Markov property. The transition function could depend on the past as other agents could learn from previous states and modify their decision function.

We propose an alternative to model the problem as an MDP that is to infer the probability state transition function from beliefs on the execution of plans as defined in Section 5.2. From belief degrees on particular actions happening, we can compute the belief degree of complete plans. Also, beliefs easily integrate other sources of information that are missing in a MDP, such as emotions or previous commitments. That is, the belief degree on an action happening may be determined, for instance, by knowing that the other agent is of a revenge type or that the other agent reached an agreement with another agent whom we trust and told us so. Moreover, as new sources of information can be easily incorporated into the world model this makes the architecture highly flexible and modular. For all those reasons we define the expected utility not for a plan in particular, but for a set of beliefs hold in a particular state as follows:

Definition 20 *We define the* expected utility *for $\alpha \in \mathcal{A}$ holding the set of* *beliefs $\mathcal{B}' \subseteq \mathcal{B}_\alpha$ in state $\omega \in W$ as:*

$$E[\mathcal{U}_\alpha(\omega, \mathcal{B}')] = \sum_{\omega_i \in W} \left(\frac{\sum_{\substack{\bar{p}\in\bar{P}^\omega \\ \mathbf{T}(\omega,\bar{p})=\omega_i}} B'(\bar{p})}{\sum_{\bar{p}\in\bar{P}^\omega} B'(\bar{p})} \times \mathcal{U}_\alpha(\omega_i) \right) \tag{5.5}$$

The previous definition does not require that α has made up his decision of what actions to perform. That is, the equation can be used at the beginning of the negotiation process when α is still uncertain on what to do, and also when all the bilateral negotiation processes have been finished and α knows what to do. The expected utility of a plan p is computed at any time assuming that the plan will be executed: $E[\mathcal{U}_\alpha(\omega, \sigma(\mathcal{B}', \{\langle\alpha, a, 1\rangle \mid a \in p\}))]$[5]. Actually, the richer the world model the more accurate the utility functions can be. We measure the level of information in a belief set as the average of Shannon s entropy among the probability distributions of actions to be operated on resources. Notice that, the higher the uncertainty the higher the entropy and thus, the less information.

Definition 21 *The uncertainty on a set of beliefs $\mathcal{B}' \subseteq \mathcal{B}_\alpha$ given the set of predicates Φ that partition it, is measured as:*

$$\mathcal{H}(\mathcal{B}') = -\frac{1}{\mid\Phi\mid}\sum_{\phi\in\Phi}\left(\frac{1}{\ln\mid\phi\mid}\sum_{\langle\alpha,\varphi_i,\vartheta_i\rangle\in\phi}\vartheta_i \ln\vartheta_i\right) \tag{5.6}$$

The uncertainty on actions to be executed is usually high at the beginning of a negotiation round, as an agent does not have enough information about each agent decisions, to 0 when the complete plan is actually performed and observed by all agents. Negotiation is the means to reduce the uncertainty on the belief model. By reaching agreements on negotiation options, agents commit to the execution of their actions in the negotiating option, and thus they reduce the uncertainty by making equal to zero the probability of executing incompatible actions on the same resource.[6]

The task of the plan search module is to find good enough plans to be executed by the agent, but also to provide good enough plans to negotiate with other agents. Plans to be executed are complete plans for the agent, that is, plans containing actions involving all the resources controlled by the agent in the current environment state. The plans to negotiate with other agents are extensions of those complete plans containing actions to be performed by other agents.

Definition 22 *Given a transition system $\Omega = \langle\mathcal{A}, R, Op, W, P, \mathbf{T}, \omega_0, W_f\rangle$ and the current state $\omega \in W$, a* joint plan *in state ω is a plan $p \in P^\omega$ involving at least two agents, $\mid\{\alpha \mid \langle\alpha, op, r\rangle \in p\}\mid \geq 2$, and complete for one of them, that is, there is $\bar{p} \in \cup_{\alpha\in\mathcal{A}}\bar{P}^\omega_\alpha$ such that $\bar{p} \subseteq p$ We denote the set of joint plans in ω by P^ω and the set of joint plans with a complete plan for α by P^ω_α.*

[5]Plans are executed at the end of the negotiation round when the certainty on the actions to be executed use to be high.

[6]If a trust model is used, and the trust in the opponent is not complete, then the belief on that agent executing an action incompatible with its commitment may not be zero. Similarly, low trust values on an agent increase the intention of not negotiating with it.

The bilateral nature of the interactions force options to contain actions to be done by, at least, two agents. In HANA, joint plans involving more than two agents can be negotiated (either concurrently or sequentially) proposing several options generated from the joint plan. In Section 5.4 we explain how options are generated from joint plans. In this section, we focus on the plan search. As introduced before, plans can be evaluated by their expected utility. Notice that this measure assumes that the plan will be executed. When joint plans are evaluated, we can also assume that the HANA agent will execute its part of the plan. Even though, we must be cautious about the actual execution of the actions in the plan assigned to other agents. We define the *confidence* of a plan in order to measure the degree of belief on the execution of a joint plan assuming that the HANA agent will perform its part of the plan.

Definition 23 *Given a set of beliefs $\mathcal{B}' \subset \mathcal{B}_\alpha$ hold by agent $\alpha \in \mathcal{A}$, and a feasible plan $p \in P^\omega$, α's* confidence *on p is defined as:*

$$\mathcal{C}_\alpha(\mathcal{B}', p) = B(\{\langle \beta, op, r\rangle \mid \langle \beta, op, r\rangle \in p \ \ and\ \beta \neq \alpha\})$$

Note that for all $\bar{p}_\alpha \in \bar{P}^\omega_\alpha$, $\mathcal{C}_\alpha(\mathcal{B}', \bar{p}_\alpha) = 1$.[7]

The output of the plan search is a *plan ranking*. The confidence measure can be used by HANA agents to rank joint plans. HANA agents can rank the joint plans by their utility (how good are they for the agent) filtering out those joint plans that do not reach a minimum level of confidence (the agent does not think that other agents will perform their actions in the plan). This minimum level of confidence may increase as time goes by and the negotiation round deadline approximates in order to focus on joint plans which confidence is high.

To generate plans we need a search algorithm that is: (i) capable to search in a huge space of solutions as required by most real scenarios, (ii) anytime as required by the time bounds, (iii) capable to generate several solutions instead of just one we are looking for several plans, and (iv) guided by a dynamic heuristic as the set of beliefs evolves with the agent interaction. We decided to implement HANA s plan generator with an evolutionary algorithm that constantly optimise the set of plans in the ranking. Concretely, we use a genetic algorithm (GA) as these algorithms allow to efficiently explore large spaces of solutions and produce successive populations of solutions that are increasingly better adapted to the environment even when it is changing along the search process. For us, each single solution, a chromosome in the GA, represents a complete or joint plan for the agent. The idea is to generate the plan ranking from the current population of solutions taking all or a subset of the best ones, preferably the latter. This population is updated generation after generation by the crossover, mutation and selection operators. It is important to guarantee the feasibility of the generated plans when applying crossover and mutation. The evaluation of a chromosome is done by the fitness function that computes the expected utility of the plan represented by the chromosome. *Fitness proportionate selection* is used to give more chances to good plans to take part of crossovers. HANA s genetic search allows to set the probability of genes being mutated. We use this to focus the search on the joint plans looking for other agent actions that can nicely extend the best complete plans. In fact, to represent plans of diverse size, we use chromosomes with size equal to the

[7]This is so because $B(\emptyset) = B(true) = 1$.

size of complete plans, and let some genes to have a void value meaning that the resource corresponding to that gene has no assigned action. The probability of void values per gene can also be adjusted. The initial population does not need to be randomised. We set the initial population and stop the search at any time saving the current population. In this way, we can resume the computation later on. It is possible to use elitism to keep the best plans alive generation after generation. Elitism in the plan generation provides a minimum of stability needed to avoid an erratic performance of the agent during negotiation. The idea is to keep the plan search running all the time, however it can be stopped and restarted if it is necessary.

Concluding, in this section we have argued the need of using data from the world model to evaluate plans. We described how plans are evaluated, and how the evaluation functions change as new observations update the world model, usually reducing the agent s uncertainty. And finally, we explained the necessary features that a plan generation should have, and how we implemented the plan generation in HANA agents to generate the ranking of plans.

5.4 Negotiation

The negotiation module uses the ranking of plans and the world model to decide how to negotiate, and what actions to perform. That is, what messages to send to other agents, and what actions to execute over the environment. The world model and the plan search have independent processes that make the world model data and the plan ranking evolve along time. The negotiation module is controlled by another process that periodically takes a snapshot of both previous modules data structure. We define this snapshot as a *negotiation state.*

Definition 24 *A* negotiation state *is a tuple*

$$s = \langle \alpha, \omega, t, \mathcal{B}^t_\alpha, \mathcal{D}^t_\alpha, \mathcal{I}^t_\alpha, P^t_\alpha \rangle$$

where:

- $\alpha \in \mathcal{A}$ *is an agent*
- $\omega \in W$ *is an environment state*
- t *is a time instant*
- $\mathcal{B}^t_\alpha$*,* $\mathcal{D}^t_\alpha$*, and* $\mathcal{I}^t_\alpha$ *are the beliefs, desires and intentions of* α *at time* t
- $P^t_\alpha : P^\omega \mapsto [0,1]$ *is a plan ranking*

We denote the set of all possible negotiation states by S. Taking a snapshot the negotiation process can use the data from the world model and the plan ranking and perform a negotiation step while the plan search is looking for even better plans[8]. The workflow of the negotiation process is as follows:

1. Takes a snapshot of the current negotiation state.
2. Generates a ranking of negotiation options.

[8]Note that the changes in the world model and the plan ranking that are done after taking the snapshot are considered in the next iteration of the negotiation process.

3. Executes the agent s intentions included in the world model.

4. Goes to (1) to continue with a new negotiation state.

The negotiation process uses the option generator to build a ranking of negotiating options satisfying the Definition 11. Options are generated from the plan ranking P^t_α as combinations of actions in joint plans $p \in P^\omega_\alpha$ that are included in the plan ranking $P^t_\alpha(p) \neq \perp$. Options are evaluated by the option evaluator that computes the next expected negotiation state assuming the acceptance of a given option $\delta \in \mathcal{O}^\omega_\alpha$. The simplest way to generate the ranking of options, $\mathcal{O}^t_\alpha : \mathcal{O}^\omega_\alpha \mapsto [0, 1]$ is as follows:

$$\mathcal{O}^t_\alpha(\delta) = f(\mathit{next}(s, \delta))$$

where $s \in S$ is the current negotiation state, $f : S \mapsto [0, 1]$ is a negotiation state evaluation function, $\mathit{next}: S \times \mathcal{O}^\omega_\alpha \mapsto S$ computes the next expected negotiation state, and $\exists\, p \in P^\omega_\alpha$ such that $\delta \subseteq p$ and $P^t_\alpha(p) \neq \perp$.

An alternative is to apply a filter and generate the option ranking using only the joint plans with value over a threshold $\nu > 0$. That is, using every plan $p \in P^\omega_\alpha$ such that $P^t_\alpha(p) > \nu$.

Given the negotiation state $s = \langle \alpha, \omega, t, \mathcal{B}^t_\alpha, \mathcal{D}^t_\alpha, \mathcal{I}^t_\alpha, P^t_\alpha \rangle$ and the option δ, an example of next expected negotiation state $\mathit{next}(s, \delta)$, for a HANA agent that always honours its commitments and fully trust the other agents, could be the negotiation state $s' = \langle \alpha, \omega, t', \mathcal{B}^{t'}_\alpha, \mathcal{D}^{t'}_\alpha, \mathcal{I}^{t'}_\alpha, P^{t'}_\alpha \rangle$ such that $\mathcal{B}^{t'}_\alpha = \sigma(\mathcal{B}^t_\alpha, \{\langle \alpha, a, 1 \rangle \mid a \in \delta\})$ and $P^{t'}_\alpha = \{p \mid p \in P^t_\alpha \text{ and } \mathrm{comp}(p, \delta)\}$. The desires and intentions would be updated according to this new set of beliefs.

HANA provides several evaluation functions for negotiation states. Other functions can be used. In general, the richer the world model the more sophisticated the evaluation functions can be. The following functions assume the basic world model with beliefs on actions.

- *Quality of information.* The higher the quality of the information that we can reach in a negotiation state the better. A well informed state contains joint plans that can reduce the uncertainty about the other agents actions. The higher the uncertainty reduction the better. A natural way to evaluate the quality of information is to define it as 1 minus the average uncertainty of Definition 21.

$$f_H(s) = \max_{p \in P^t_\alpha} (1 - \mathcal{H}(\sigma(\mathcal{B}^t_\alpha, \{\langle \alpha, a, 1 \rangle \mid a \in p\}))) \tag{5.7}$$

- *Independence.* The more independent an agent is the better. If an agent can reach a high utility by its own means the better the negotiation state is. This measure depends on the complete plans for the agent that have been found so far, $\bar{P}^t_\alpha = \{p \mid p \in \bar{P}_\alpha \text{ and } P^t_\alpha(p) \neq \perp\}$. A state is as good as the best state the agent can reach by its own means, this is the maximum expected utility to be obtained by assuming we choose one of the complete plans for agent α:

$$f_{\mathcal{U}C}(s) = \max_{\bar{p} \in \bar{P}^t_\alpha} E[\mathcal{U}_\alpha(\omega, \sigma(\mathcal{B}^t_\alpha, \{\langle \alpha, a, 1 \rangle \mid a \in \bar{p}\}))] \tag{5.8}$$

- *Opportunity.* The more utility to be obtained with joint plans the better. Finding joint plans that give high utility is actually the reason of the whole negotiation process. Any state that has joint plans with high expected utility is a good state. This measure is similar to the previous one but using joint plans $P^t_\alpha = \{p \mid p \in P_\alpha \text{ and } P^t_\alpha(p) \neq \bot\}$:

$$f_{\mathcal{U}J}(s) = \max_{\hat{p} \in \hat{P}^t_\alpha} E[\mathcal{U}_\alpha(\omega, \sigma(\mathcal{B}^t_\alpha, \{\langle \alpha, a, 1\rangle \mid a \in p\}))] \tag{5.9}$$

- *Confidence.* The more confidence in the available plans the better. Having a high confidence in the plans found during the search the less uncertainty on what will happen.

$$f_{\mathcal{C}}(s) = \max_{p \in P^t_\alpha} \{\mathcal{C}_\alpha(\mathcal{B}^t_\alpha, p) \mid p \in P^\omega, P^t_\alpha(p) \neq \bot\} \tag{5.10}$$

Each of these different measures, or a combination of them, allows to evaluate negotiation states and thus rank the available options. When to use each measure, and how to combine them is what determines an agent s negotiation strategy. HANA allows to define strategies combining these measures. The other key element of the negotiation strategy is the *aspiration level*, i.e. the minimum evaluation value that the agent has for options to be acceptable. The options above the aspiration level should be accepted. Otherwise, rejected. At the beginning of a negotiation round, agents would usually request a high aspiration value. As time goes by and the deadline approaches, agents become less demanding, and they decrease their expectations in order to reach some agreements that improve, even in a low amount, their negotiation state. HANA allows to define the way the aspiration level decreases as the next definition shows.

Definition 25 *Given a negotiation state s, a deadline t_{max}, and current time t, the* aspiration level*, denoted $\mathcal{A}(s,t)$, is defined as:*

$$\mathcal{A}(s,t) = g_{min}(s,t) + \left(\frac{t_{max} - t}{t_{max}}\right)^\tau \cdot (1 - g_{min}(s,t))$$

where $\tau \in [0,1]$ is the aspiration decay rate and $g_{min}(s,t)$ is the minimum value that can be guaranteed.

The negotiation strategy is then determined by fixing values for $g_{min}(s,t)$. HANA allows to define these functions as linear combinations of the measures defined before. That is,

$$g_{min}(s,t) = w_1(t) \cdot g_1(s) + w_2(t) \cdot g_2(s) + ... + w_n(t) \cdot g_n(s)$$

where every $g_i(s) \in [0,1]$ is a measure over the state s and $\sum_{i<n} w_i(t) = 1$. Next we discuss a few negotiation strategies:

- *Conservative.* An agent can guarantee a minimum utilitarian value with its own actions that corresponds to $g_{min}(s) = f_{\mathcal{U}C}(s)$. This strategy is convenient at the end of a negotiation round as it concedes maximally towards the guaranteed minimum.

- *Informative.* A convenient strategy at the beginning of a negotiation round is to increase the agent s information quality. This facilitates to explore the space of options and reduce uncertainty in the negotiation. The more information an agent has the more probable its future proposals will be accepted. This can be achieved by $g_{min}(s) = f_H(s)$.

- *Dynamic.* A combination of the previous two strategies starting with informative and ending with conservative.

$$g_{min}(s,t) = w_1(t) \cdot f_H(s) + w_2(t) \cdot f_{\mathcal{U}C}(s)$$

  where $w_1(t) = \frac{t_{max}-t}{t_{max}}$ and $w_2(t) = 1 - w_1(t)$.

As introduced in Section 5.2, the HANA agents have three basic intentions that are: *reply* δ, *propose* and *executeActions*. Desires and intentions are graded and are represented similarly to how beliefs are represented. The basic intentions are mutually incompatible, thus the aggregation of their degrees is always 1. HANA agents satisfy the negotiation protocol defined in Section 3.2 providing transition functions from beliefs and desires to intentions. Those transition functions update the degrees of the basic intentions whenever a message is sent, or the deadline is reached. The negotiation process executes the current basic intention with the highest degree.

At the beginning of the negotiation round, the highest basic intention is always to *propose*: $\mathcal{I}^t_\alpha(\text{propose}) > \mathcal{I}^t_\alpha(\text{reply } \delta) \wedge \mathcal{I}^t_\alpha(\text{propose}) > \mathcal{I}^t_\alpha(\text{executeActions})$. That is also the case when there is no proposal received, and there is time left before the timeout. Whenever the highest basic intention is *propose*, the negotiation process executes the following sentences proposing the best option only when it overcomes the current aspiration of the agent:

```
**Ensure:** $s$ is the current negotiation state
  $\delta \leftarrow \arg\max_{\delta \in \mathcal{O}^\omega_\alpha} \mathcal{O}^t_\alpha(\delta)$ {selects the best ranked option}
  **if** $\mathcal{O}^t_\alpha(\delta) > \mathcal{A}(s,t)$ **then**
    propose($\delta$) {the interface module sends a message proposing $\delta$}
  **end if**
```

When a proposal is received, the highest basic intention becomes to be *reply* δ, that is to reply the received proposal on the negotiating option δ. Then, the negotiation process accepts the proposal only when the next expected state provided by δ has a better evaluation than the current aspiration of the agent:

```
**Ensure:** $f : S \mapsto [0,1]$ is the negotiation state evaluation function
**Ensure:** $s$ is the current negotiation state
  $s' \leftarrow next(s,\delta)$ {computes the next expected state}
  **if** $f(s') > \mathcal{A}(s,t)$ **then**
    accept($\delta$) {the interface module sends a message accepting $\delta$}
  **else**
    reject($\delta$) {the interface module sends a message rejecting $\delta$}
  **end if**
```

Finally, when the timeout is reached, the highest basic intention is to *executeActions*. Then, the negotiation protocol automatically cancel all ongoing negotiations and the HANA negotiation process selects the best complete plan from the plan ranking, and executes it:

```
$\bar{P}^t_\alpha \leftarrow \{p \mid p \in \bar{P}^\omega_\alpha$ and $P^t_\alpha(p) \neq \perp\}$
$\bar{p} \leftarrow \arg\max_{\bar{p} \in \bar{P}^t_\alpha} E[\mathcal{U}_\alpha(\omega, \sigma(\mathcal{B}^t_\alpha, \{\langle \alpha, a, 1 \rangle \mid a \in \bar{p}\})$ {available complete plan with highest utility}
execute($\bar{p}$) {the interface module executes all actions in $\bar{p}$}
```

Other intentions can be used by the negotiation strategy when desired. For example, a HANA agent with a trust model incorporated can generate the intention to *not negotiate with agent β*. In that case, the previous algorithm should be modified to reject any proposal from β. And the plan evaluation of the plan search module should be modified to poorly evaluate joint plans with plans including β. The HANA architecture is designed to be able to incorporate new models in the world module and adjust the rest of the components. The architecture is flexible and allows to create a high diversity of different agents.

5.5 Summary

The HANA agent architecture is composed by four modules: the interface, the world model, the plan search and the negotiation. The interface contains the sensors and actuators of HANA agents. It perceives the environment state, performed actions, and received messages. It also perceives the time. The agent actuators can send messages and perform actions, however, the decision making is not included in the interface module.

Everything that is observed is stored as beliefs in the world model. Beliefs are graded as well as desires and intentions. The degree represents the level of confidence that the agent has on each belief, desire or intention. We model those degrees as probabilities because the data comes from the previous interactions with the agents and thus, can be statistically processed. Therefore, the conjunction of beliefs is computed as the product of their degrees. Beliefs and desires generate intentions together. Intentions are not necessarily actions or plans to perform. They use to be general instructions like the level of trust to use for a given agent, or the level of gain requested for deals to be accepted. The nature of the beliefs, desires and intentions is given by the behavioural models that can be plugged into the world model. The architecture has been designed in order to be extensible incorporating other models compatible with a BDI architecture like HANA. Another peculiarity of this world model is that desires are not fixed. They evolve triggered by the agent s emotion that change due to observations processed as beliefs.

To cope with the huge space of possible solutions, HANA uses a plan search that consists of an anytime algorithm that generates and upgrades a ranking of complete and joint plans for the agent. This generator generates plans for a given environment state, and includes into the ranking only those that are good enough according to a plan evaluator that makes use of the information stored in the world model. This evaluator is a first level filter. A measure of the expected utility, the confidence and the uncertainty that it can provide are used to evaluate those plans.

Joint plans included in the plan ranking are used to generate the ranking of negotiation options. Options are evaluated taking into account the world model. Several measures can be used to evaluate those options. Even though, the final decision about what actions to perform, what options to propose or accept is taken by the negotiation strategy. The negotiation strategy has access to the ranking of complete plans, the ranking of options and the time. The strategy is guided by the intentions of the agent adjusting its parameters to the current intentions. We provide a branch of possible strategies that can be combined taking into account the current negotiation state and the time. The idea is that every HANA agent has its own negotiation strategy and evaluation functions. And that they could make use of extra information provided by incorporated behavioral models without having to deal with the search, the communication or any model that was previously incorporated. The creation of agents is thus done as a combination of pieces improving the previous agents with new capabilities.

Chapter 6

DipGame testbed

The DipGame testbed is the infrastructure for software agents to be created, and experiments to be executed using The Diplomacy Game as specific domain for RNP and similar problems. The chapter starts in Section 6.1 motivating the use of Diplomacy for research purposes. In Section 6.2, we describe the infrastructure, and in Section 6.3 the language. The infrastructure consists of several components to assist the creation of bots, described in Section 6.4, and others to assist the execution of experiments and their analysis, described in Section 6.5. An experiment proving the suitability of the testbed for evaluating negotiating agents is included in Section 6.6. We conclude the chapter with a summary of the content, Section 6.7.

6.1 Diplomacy for research

The Diplomacy Game is a strategically simple game for humans to play in comparison to other classic games like Chess or Go. This is because the true complexity of the game lies in the management of the relationships among players. Relationships are subtle, constantly changing and may appear at first sight difficult to analyse. However, humans often negotiate in their everyday life, and they are used to manage relationships. They are very used to do it and thus use to see it as a quite trivial task. Diplomacy is thus not perceived by humans as a difficult game to play, but it is really difficult for a computer as the search space is huge, and the key for success relies on the information obtained from negotiation rounds, the ability of agents to build relationships and on the persuasive capability of the player. Chess and Go are solved. There are programs capable to defeat experts of those games. However, games that involve inter-player relationships are not.

Focusing just on the possible moves that the units can perform, the combinations are very large. We studied the complexity of the game in Section 4.7 and ended up with a branching factor of $15^{30} = 1.43 \cdot 10^{22}$ for a standard no-press (without negotiation) game without convoys. This complexity cannot be treated by standard search mechanisms. Think that the branching factor for Chess and Go are around 35 and 200 respectively [Burmeister and Wiles, 1995]. Therefore, we cannot take advantage of the kinds of massive parallel computation that are available in algorithms for playing them. To solve The Diplomacy

Game problem, we must understand the relationships between players, their reaction and take advantage of all information gathered from negotiation. All this makes The Diplomacy Game a very interesting game for research.

6.2 Infrastructure

The Diplomacy Game is very popular and, as introduced in Section 2.4, there is previous work on creating software agents capable to play Diplomacy. The first bots had few in common [Kraus et al., 1989; Loeb and Hall, 1990] but since the emergence of DAIDE, see Section 2.4.2, it has been used in most of the works. DipGame is not an exception. In fact, DipGame can be seen as the adaptation of DAIDE to the MAS research community and AI in general.

DAIDE establishes a Client-Server communication model, see Figure 6.1. The server is the game manager, that is, the software that takes all the orders and runs the adjudicator algorithm, see Chapter 4, to generate the next turn. All clients are connected directly to the server. They can never communicate directly with other clients. All communications go through the server. Clients are any piece of software that is observing the game. Clients can be classified as *players* and *observers*. The obvious difference between them is that players can play while observers do only observe the game, that is: to receive information on the game progress. Players, in addition, must send their orders to the server for the game to proceed with the next turn. Figure 6.2 illustrates a possible distribution of players and observers for a given game. Remember that seven players are required for a Diplomacy game to start.

The communication between clients and the server is done via TCP/IP following the protocol defined in [Rose, 2003]. This allows to set games where the server and the clients do not need to be running on the same computer. They can run on different computers connected to the same network, e.g. Internet. In fact, they do not even need to use the same programming language. They just need to follow the protocol that specifies how data must be transferred. The messages that client and server can exchange follow the language defined in [Norman, 2006] as a hierarchy. There are fourteen language levels denoted by Level X where X can be one of the following numbers: 0, 10, 20, 30, 40, ..., and 130. The first level, Level 0, is the *no-press* level. It contains all the terms needed to run a game without negotiation. This level is used by many bots. Contrarily, the upper levels, 10 130, are used only by a couple of bots that are those that can negotiate to some extend [Kemmerling et al., 2012; van Hal]. They achieve only level 20 and even with some missing terms. The language was defined specifically for The Diplomacy Game and do not contemplate some of the features necessary for using the game in research. For instance, the domain dependent terms are coupled to the language itself. We define a new language denoted by L that separates the Diplomacy vocabulary from the language and provides a division in several levels of complexity that go in line with the MAS research topics. If the *no-press* part of the game is formalised in Chapter 4, the negotiation part is formally defined in Section 6.3.

In addition to the communication standard, DAIDE provides two software tools developed by David Norman: those are the *AiServer*[1] a DAIDE server, and AiMapper a DAIDE client for humans to play and observe games where

[1]`http://www.ellought.demon.co.uk/dipai/`.

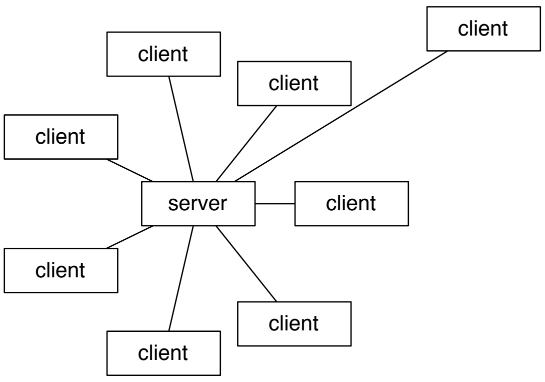


Figure 6.1: Example of DAIDE s client-server communication model.

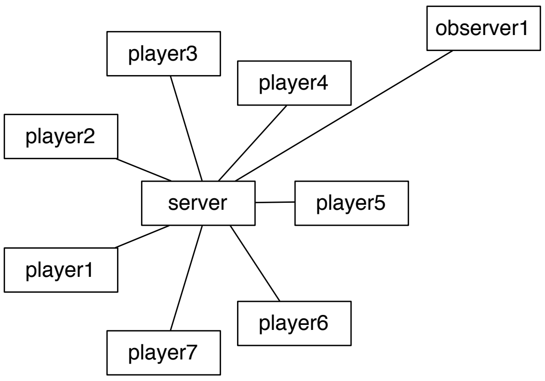


Figure 6.2: Example of player and observer distribution for a game.

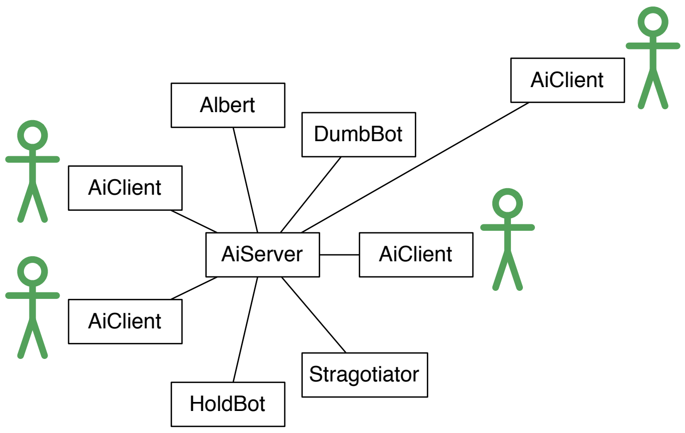


Figure 6.3: Example of DAIDE clients that can be used for the distribution at Figure 6.2. Three players are humans using AiClient, the other human is an observer using the same software but in different mode. The rest are bots.

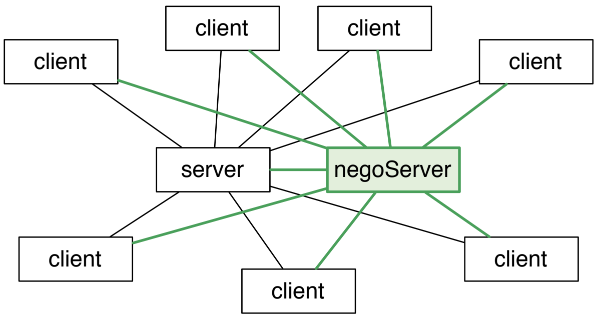


Figure 6.4: DipGame communication model.

orders are illustrated on a map. These clients with a map are called mappers. Both tools are very useful, unfortunately they run only on Windows. An alternative multi-platform server is *Parlance*,[2] developed by Eric Wald. Although there are no many servers and mappers, there are several bot development frameworks for C/C++ and .Net, and a communication API for Java. All of them provide a library for the TCP/IP communication following the communication protocol. But only the frameworks provide a bot skeleton and a representation of the game state including the current unit positions and performed orders. The most known bots implemented using these frameworks or the Java API are: HoldBot, Randbot, DumbBot, BlabBot, HaAI, KissMyBot, Albert, Diplominator, Brutus, and Stragotiator. Figure 6.3 illustrate an example of game with several human players and some of these bots.

In DipGame we refuse to reinvent the wheel and use the AiServer, AiMapper and Parlance for running experiments, see sections 6.5.2 and 6.5.1. And we build a bot development framework (*dip*) that makes use of a reviewed version of the Java communication API as being multi-platform is a recommended feature for our testbed. See Section 6.4.1 for a detailed description of *dip*.

In addition, it provides a parallel server for negotiation and a negotiation library to allow negotiation processes to pass through this server using L instead of the DAIDE server. Figure 6.4 illustrates the DipGame communication module where all clients are also connected to the *negoServer* and it is connected to the DAIDE server as an observer.

As many programs must be executed in order to run a simple game, we provide the *GameManager* that is a software that co-ordinates all of them, allows to configure the games and collects all data. This data can be analysed with *DipTools* that is the last component of our testbed. Sections 6.5.2 and 6.5.3 describe these two components of DipGame.

The infrastructure is multi-platform and modular. It is freely available at `http://www.dipgame.org` together with examples of clients and more documentation.

[2]`http://pypi.python.org/pypi/Parlance`.

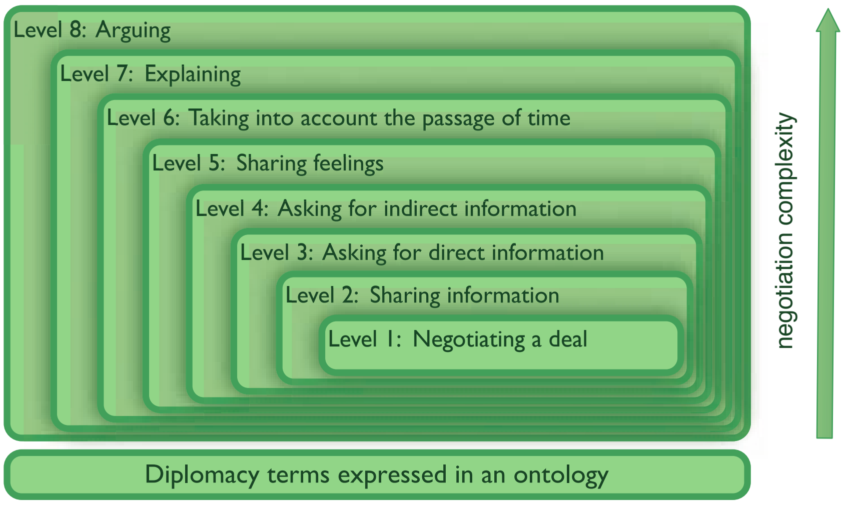

Level 8: Arguing
Level 7: Explaining
Level 6: Taking into account the passage of time
Level 5: Sharing feelings
Level 4: Asking for indirect information
Level 3: Asking for direct information
Level 2: Sharing information
Level 1: Negotiating a deal
Diplomacy terms expressed in an ontology
negotiation complexity

Figure 6.5: Language hierarchy. Each language L_i extends the languages in lower levels, that is, if there is no re-writing rule for a term in L_i then it can be found in lower levels L_j, with $j < i$.

6.3 Negotiation language

The complexity of building an agent capable to negotiate is correlated to the complexity of the language that the agent must be able to understand. The higher the language complexity the higher the richness of the models underpinning agent architectures. In this section we structure the expressions of increasing levels of complexity via a modular, flexible and reusable language hierarchy L. We propose it as a standard for the dialogical communication between the agents that use the testbed. Nonetheless, other languages could be used as the testbed is quite modular and the language tools (i.e. parsers) are separated from the game engine.

Figure 6.5 graphically represents the language hierarchy L that is defined as an eight level hierarchy; starting from L_1 and increasing the expressiveness as the language level increases. Check the definition of L in Figure 6.6. The higher the language level that we use, the more complex the actions and the predicates, and thus the expressivity. If it is desired, some of the levels could be skiped, such as for instance Level 5 of sharing feelings. By this way, you can reach level 8 corresponding to argumentation without expressing any feelings. We kept a linear approach for two reasons. One for simplicity, as it gives you a clear roadmap to building ever more complex negotiation agents. Second, to follow the familiar level ordering in the languages proposed by the DAIDE community. Researchers set their experiments on DipGame selecting the language level to use.

L is a generic language that could be used for many other applications. It defines the illocutions that the agents can use to communicate and the basic concepts like *Agree*, *Desire*, *Feel*, etc. The language is parametric on the vocabulary for a specific application domain, described as an ontology. The undefined

Level 1: Negotiating a deal
$L_1 ::= \text{propose}(\alpha, \beta, \mathit{deal}_1) \mid \text{accept}(\alpha, \beta, \mathit{deal}_1) \mid \text{reject}(\alpha, \beta, \mathit{deal}_1) \mid \text{withdraw}(\alpha, \beta)$
$\mathit{deal}_1 ::= \text{Commit}(\alpha, \beta, \varphi)^+ \mid \text{Agree}(\beta, \varphi)$
$\varphi ::= \underline{\mathit{predicate}} \mid \text{Do}(\underline{\mathit{action}}) \mid \varphi \wedge \varphi \mid \neg\varphi$
$\beta ::= \alpha^+$
$\alpha ::= \underline{\mathit{agent}}$

Level 2: Sharing information
$L_2 ::= L_1 \mid \text{inform}(\alpha, \beta, \mathit{info}_2)$
$\mathit{info}_2 ::= \mathit{deal}_1 \mid \text{Obs}(\alpha, \beta, \varphi) \mid \text{Belief}(\alpha, \varphi) \mid \text{Desire}(\alpha, \varphi) \mid \mathit{info}_2 \wedge \mathit{info}_2 \mid \neg \mathit{info}_2$

Level 3: Asking for direct information
$L_3 ::= L_2 \mid \text{inform}(\alpha, \beta, \mathit{info}_3) \mid \text{query}(\alpha, \beta, \mathit{info}_3) \mid \text{answer}(\alpha, \beta, \mathit{info}_3)$
$\mathit{info}_3 ::= \mathit{info}_2 \mid \text{Unknown}(\alpha, \mathit{info}_3) \mid \mathit{info}_3 \wedge \mathit{info}_3 \mid \neg \mathit{info}_3$

Level 4: Asking for indirect information
$L_4 ::= L_3 \mid \text{inform}(\alpha, \beta, \mathit{info}_4) \mid \text{query}(\alpha, \beta, \mathit{info}_4) \mid \text{answer}(\alpha, \beta, \mathit{info}_4) \mid \text{inform}(\alpha, \beta, L_4) \mid \text{query}(\alpha, \beta, L_4) \mid \text{answer}(\alpha, \beta, L_4)$
$\mathit{info}_4 ::= \mathit{info}_3 \mid \text{Unknown}(\alpha, \mathit{info}_4) \mid \text{Unknown}(\alpha, L_4) \mid \mathit{info}_4 \wedge \mathit{info}_4 \mid \neg \mathit{info}_4$

Level 5: Sharing feelings
$L_5 ::= L_4 \mid \text{inform}(\alpha, \beta, \mathit{info}_5) \mid \text{query}(\alpha, \beta, \mathit{info}_5) \mid \text{answer}(\alpha, \beta, \mathit{info}_5) \mid \text{inform}(\alpha, \beta, L_5) \mid \text{query}(\alpha, \beta, L_5) \mid \text{answer}(\alpha, \beta, L_5)$
$\mathit{info}_5 ::= \mathit{info}_4 \mid \text{Unknown}(\alpha, \mathit{info}_5) \mid \text{Unknown}(\alpha, L_5) \mid \text{Feel}(\alpha, \mathit{feeling}) \mid \mathit{info}_5 \wedge \mathit{info}_5 \mid \neg \mathit{info}_5$
$\mathit{feeling} ::= \mathit{VeryHappy} \mid \mathit{Happy} \mid \mathit{Sad} \mid \mathit{Angry}$

Level 6: Taking into account the passage of time
$L_6 ::= L_5 \mid \text{propose}(\alpha, \beta, \mathit{deal}_6, t) \mid \text{accept}(\alpha, \beta, \mathit{deal}_6, t) \mid \text{reject}(\alpha, \beta, \mathit{deal}_6, t) \mid \text{withdraw}(\alpha, \beta, t) \mid \text{inform}(\alpha, \beta, \mathit{info}_6, t) \mid \text{query}(\alpha, \beta, \mathit{info}_6, t) \mid \text{answer}(\alpha, \beta, \mathit{info}_6, t) \mid \text{inform}(\alpha, \beta, L_6, t) \mid \text{query}(\alpha, \beta, L_6, t) \mid \text{answer}(\alpha, \beta, L_6, t)$
$\mathit{info}_6 ::= \mathit{info}_5 \mid \mathit{deal}_6 \mid \text{Obs}(\alpha, \beta, \varphi_6, t) \mid \text{Belief}(\alpha, \varphi_6, t) \mid \text{Desire}(\alpha, \varphi_6, t) \mid \text{Unknown}(\alpha, \mathit{info}_6, t) \mid \text{Unknown}(\alpha, L_6, t) \mid \text{Feel}(\alpha, \mathit{feeling}, t) \mid \mathit{info}_6 \wedge \mathit{info}_6 \mid \neg \mathit{info}_6$
$\mathit{deal}_6 ::= \mathit{deal}_5 \mid \text{Commit}(\alpha, \beta, \varphi_6, t)^+ \mid \text{Agree}(\beta, \varphi_6, t)$
$\varphi_6 ::= \underline{\mathit{predicate}} \mid \text{Do}(\underline{\mathit{action}}, t) \mid \varphi_6 \wedge \varphi_6 \mid \neg\varphi_6 \mid \varphi_6; \varphi_6$
$t ::= \underline{\mathit{time}}$

Level 7: Explaining
$L_7 ::= L_6 \mid \text{inform}(\alpha, \beta, \mathit{info}_7, t) \mid \text{query}(\alpha, \beta, \mathit{info}_7, t) \mid \text{answer}(\alpha, \beta, \mathit{info}_7, t) \mid \text{inform}(\alpha, \beta, L_7, t) \mid \text{query}(\alpha, \beta, L_7, t) \mid \text{answer}(\alpha, \beta, L_7, t)$
$\mathit{info}_7 ::= \mathit{info}_6 \mid \text{Unknown}(\alpha, \mathit{info}_7, t) \mid \text{Unknown}(\alpha, L_7, t) \mid \text{Explain}(\mathit{info}_7, t) \mid \text{Explain}(L_7, t) \mid \mathit{info}_7 \wedge \mathit{info}_7 \mid \neg \mathit{info}_7$

Level 8: Arguing
$L_8 ::= L_7 \mid \text{inform}(\alpha, \beta, \mathit{info}_8, t) \mid \text{query}(\alpha, \beta, \mathit{info}_8, t) \mid$
$\text{answer}(\alpha, \beta, \mathit{info}_8, t) \mid \text{inform}(\alpha, \beta, L_8, t) \mid \text{query}(\alpha, \beta, L_8, t) \mid$
$\text{answer}(\alpha, \beta, L_8, t)$
$\mathit{info}_8 ::= \mathit{info}_7 \mid \text{Unknown}(\alpha, \mathit{info}_8, t) \mid \text{Unknown}(\alpha, L_8, t) \mid$
$\text{Explain}(\mathit{info}_8, t) \mid \text{Explain}(L_8, t) \mid \text{Attack}(\mathit{info}_7, \mathit{info}_7) \mid$
$\text{Support}(\mathit{info}_7, \mathit{info}_7) \mid \mathit{info}_8 \wedge \mathit{info}_8 \mid \neg \mathit{info}_8$

Figure 6.6: Language hierarchy definition in Backus-Naur Form. Note that: *expression*$^+$ denotes a non-empty sequence of *expression*, non terminal symbols are written in italic, and undefined symbols (referring to terms in the ontology) appear in underlined italics.

non terminal symbols that appear in L should be specifically defined for each application domain. These symbols are: *time*, *agent*, *action* and *predicate*. Figure 6.7 a) represents the ontology for the Diplomacy game and Figure 6.7 b) contains the connexion between the ontology and L. In this way, we allow researchers to reuse as much code as possible when applying their work to real world applications after testing it with DipGame.

year ::= <u>integer</u>
season ::= spr | sum | fal | aut | win
power ::= fra | eng | tur | rus | ita | aus | ger
coast ::= ncs | scs | ecs | wcs
regionType ::= amy | sea | coast
supplyCenter ::= spa | mar | par | stp | ...
province ::= *supplyCenter* | gas | bur | sil | tus | ...
region ::= Region(*province*, *regionType*)
unit ::= Unit(*power*, *region*)
order ::= hld(*unit*) | mto(*unit*, *region*) | sup(*unit*, hld(*unit*)) | sup(*unit*, mto(*unit*, *region*)) | rto(*unit*, *region*) | dsb(*unit*) | bld(*unit*) | rem(*unit*) | wve(*power*)
offer ::= pce($power^+$) | aly($power^+$, $power^+$)

a) Ontology

agent ::= *power*
action ::= *order*
predicate ::= *offer*
time ::= ⟨*season*, *year*⟩

b) Connexion

Figure 6.7: Diplomacy ontology and connexion to L.

In the rest of this section, we describe and illustrate the expressivity of each language level in Diplomacy. For instance, `Unit(rus, Region(stp, scs))` is a term meaning that `There is a unit from Russia in the south coast of Saint Petersburg , `pce([ita rus])` is a predicate meaning `Peace between Italy and Russia , and `sup(Unit(rus, Region(spa, ecs)), mto(Unit(ita, Region(mar, amy)), Region(par, amy)))` is an example of action where `The unit of Russia in the east coast of Spain supports the movement of the Italian army in Marseilles to Paris .

L_1: **Negotiating a deal.** This is the first language level. It allows agents to negotiate deals. The deals can be either a sequence of commitments, one for every agent involved in the deal, or a global agreement in which a set of agents agree on something, usually the truth of a predicate. Here you have two examples of sentences in L_1: E.g. `Italy proposes to Russia a deal by which Italy commits to do a movement from its army in Marseilles to Paris and Russia commits to support the Marseilles italian army s movement with the unit in the east coast of Spain :[3]

```
propose(ita, rus,
   [Commit(ita,rus,
        Do(mto(Unit(ita, Region(mar, amy)),
                Region(par, amy))))
   Commit(rus, ita,
        Do(sup(Unit(rus, Region(spa, ecs)),
                mto(Unit(ita, Region(mar, amy)),
                    Region(par, amy)))))])
```

E.g. `Italy accepts to Agree with Russia that they are allied against England :

[3]We abuse notation and represent sets of agents containing just one agent by the agent name itself. e.g. Commit(`ita`, [`rus`], *deal*) will be represented as Commit(`ita`, `rus`, *deal*).

accept(`ita`, `rus`, Agree([`ita rus`], `aly`([`ita rus`], `eng`)))

L_2**: Sharing information.** This language level adds the ability of sharing information with other agents. It can be information about previous commitments, observed actions, beliefs, desires or deals. E.g. `Italy informs England that Italy keeps a peace agreement with Russia :

inform(`ita`, `eng`, Agree([`ita rus`], `pce`([`ita rus`])))

L_3**: Asking for direct information.** At level three, agents can request other agents for information. Answers to queries are similar to informs. E.g. `England asks Italy whether Italy and Russia have a peace agreement :

query(`eng`, `ita`, Agree([`ita rus`], `pce`([`ita rus`])))

E.g. `Italy answers England that Italy and Russia do have a peace agreement :

answer(`ita`, `eng`, Agree([`ita rus`], `pce`([`ita rus`])))

L_4**: Asking for indirect information.** Level four allows to inform about dialogical moves between agents. E.g. `Russia asks Italy whether Italy answered to England that Italy and Russia had a peace agreement :

```
query(rus, ita,
      answer(ita, eng,
             Agree([ita rus], pce([ita rus]))))
```

L_5**: Sharing feelings.** This level is the emotional one. Feelings can be exchanged between agents. E.g. `Italy asks Russia whether Italy s answer to England that Italy and Russia had a peace agreement made Russia feel sad :

```
query(ita, rus,
      answer(ita, eng,
             Agree([ita rus], pce([ita rus]))) →
      Feel(rus, Sad))
```

L_6**: Taking into account the passage of time.** L_6 adds time to L_5. Within L_6 we can speak about the past and make promises about the future. Time is added as an extra argument to predicates and illocutions. Time variables are considered universally quantified. In subsequent levels, L_7 and L_8, we omit time to simplify notation. E.g. `Russia informs Italy that if Italy informs in the future to any power that Italy and Russia have a peace agreement, then Russia will feel Angry :

```
inform(rus, ita,
       (inform(ita, power,
               Agree([ita rus],
                     pce([ita rus])), $t_1$) ∧ $t_1 > t_0$) →
       (Feel(rus, Angry, $t_2$) ∧ $t_2 > t_1$),
       $t_0$)
```

L_7**: Explaining.** Dialogues often include explanations and explanation requests. This level adds that possibility to allow agents to explain why things are like they are. E.g. `Italy asks Russia for an explanation of why the fact that Turkey believes that there is a peace agreement between Russia and Italy makes Russia feel Angry :

query(`ita`, `rus`,
  Explain(
    Belief(`tur`,
      Agree([`ita rus`], `pce`([`ita rus`]))) $\rightarrow$
    Feel(`rus`, Angry)))

L_8**: Arguing.** And finally, level 8 allows agents to express rebuttals and supports between arguments. E.g. `Russia informs England that its alliance with Italy against England and Italy s desire to conquer Paris together support the imminent Italian attack from Marseilles to Paris :

inform(`rus`, `eng`, Support(
  Agree([`ita rus`],`aly`([`ita rus`],`eng`)) $\wedge$ Desire(`ita`, `par`),
  Do(`mto`(`Unit`(`ita`, `Region`(`mar`, `amy`)), `Region`(`par`, `amy`)))))

6.4 Creating a bot

In this section we describe our bot development framework and the negotiation utilities that we provide in *nego*. Then, we define a methodology for building bots using DipGame and we exemplify the usage of the testbed with *Random-NegoBot* in Section 6.4.3.

6.4.1 dip

Developing a client from scratch is tedious. That is why we provide *dip*, a java framework that copes with the communication with the game manager and the representation of the game state and the movements (referred in the game as orders to send to the units). The *dip* framework is designed with a separate module to deal with DAIDE s communication protocol. This module is called *daideComm* and uses the Java Communication API (*jac*) to encode messages and connect to DAIDE servers. The separation of this module is recommended in order to be able to easily adapt the testbed to run using other communication protocols. For example, to run on Droidippy, see Chapter 7.

dip is very easy to use, for instance, creating a player means just to implement a new class extending the abstract class *Player* (see Figure 6.8) and adding the extra functionality to decide what movements to do next, as explained in Figure 6.9. The rest of the work is done by the framework itself. To illustrate how to use *dip* we provide *ConsoleObserver* and *ConsolePlayer* that are console applications that allow a user to observe a game and play respectively. Those applications have no graphical user interface.

Deciding what movement to do next usually requires to search the space of possible actions to perform. For those researchers that want to test their work and are not interested in the search process, we provide an extension of *dip*, called *bot*, that calculates the potential actions that the bot may choose based on an action evaluation function that the researcher defines. This simplifies even more the implementation of a player because it then consists basically in defining an evaluation function. This function can be defined as a combination of the implementation of three class interfaces: *RegionEvaluator*, *OrderEvaluator* and *OptionEvaluator*.[4] Each class interface is in fact an evaluation function itself

[4]An option in this context is the set of all orders to be performed by the bot.

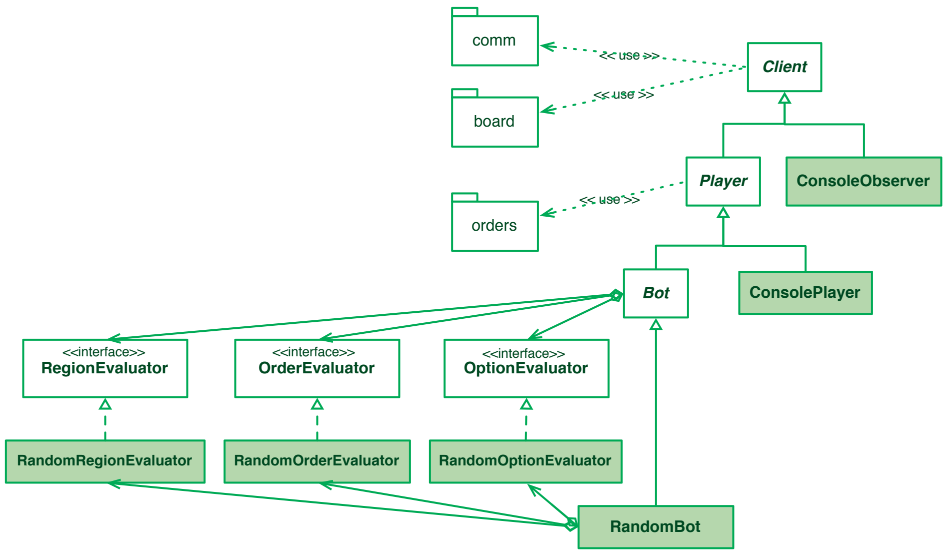


Figure 6.8: UML 2.0 class diagram of the most important classes in the framework.

Client type	Plays?	Provided funct.	Required funct.	Required library	Objects to implement	Implement. example
observer	no	game state	nothing	dip	Observer	ConsoleObserver
player	yes	game state	action selection	dip	Player	ConsolePlayer
player	yes	game state, best action	action evaluation	dip+bot	Bot RegionEvaluator OrderEvaluator OptionEvaluator	RandomBot

Figure 6.9: Bot development framework.

over a different dimension (region, order and option). The combination of them is thus not optimal (as it considers them in isolation) but is a good trade-off between memory usage and solution quality. The trade-off level is fixed by the programer: the more memory the higher the solution quality. An example of bot implemented using *bot* is *RandomBot*[5] whose evaluation function always returns 0. Figures 6.8 and 6.9 summarize the content and use of the infrastructure.

6.4.2 nego

The *nego* package provides support for the negotiation between players. We take the language L as a standard for this testbed and provide a parsing utility called *dipNego* that checks the syntax of messages and represents them in a structure of objects. In Figure 6.10 we illustrate a class diagram with the most relevant classes of *dipNego* that are necessary to represent the messages in L_1. We can parse all L language levels and the Diplomacy ontology. In the figure we represent the language and the ontology classes together, e.g. `Peace` and `Alliance`. Nonetheless, as L is domain independent, *dipNego* can be used for

[5]http://www.dipgame.org/browse/examples

other application domains. The `Message` class is the wrapper for an illocution to be communicated between two or more agents. `DealIllocutions` refer to offers and `Do` allows to specify actions. In the context of Diplomacy an action is an order.

The negotiation dialogues between players are handled independently from the game engine. Negotiating messages do not use DAIDE s protocol and do not pass through the DAIDE server. Instead, players negotiate using *negoServer*, an instant messaging program specially created for this testbed. The *negoClient* library implements the functionality required to connect a client with the negotiation server. Therefore, to add negotiation capability to an existing bot, we may use *negoClient* in addition to the *dipNego* library.

In order to be able to play against human agents DipGame provides a library that translates human messages about Diplomacy from (a restricted set of) English into the formal language that *dipNego* supports. This library is called *dialogueAssistant* and makes use of previous exchanged messages, the dialogues, to interpret the new ones. Figure 6.11 illustrates the use of the library.

DipGame do not constrain the negotiation to a specific protocol. Even though a protocol can be imposed like for example the one illustrated in Figure 6.12 where every proposal must be replied and information can be shared and queries can be formulated at any time during a negotiation process. As the negotiation goes in parallel to the game and through the *negoServer*, we can control the language and protocol and set it to the needs of the given experiment.

6.4.3 Methodology

Building agents capable to negotiate playing Diplomacy is quite easy with the framework and the tools introduced above because the communication problems are fixed and the developer can focus on the negotiation model of its agents. We suggest a methodology for creating agents with DipGame and exemplify it describing how can we create a very simple agent: a random negotiator, that is, a player that performs random moves and negotiates also randomly. We base it on *RandomBot* s code.[6]

The methodology is quite simple. It consists of six steps that end with the analysis of the results and the optional improvement of the agent:

1. Download all the required resources. They will depend on the type of client that you want to create and the implementation options that you decide to use, see Figure 6.9.

   For our example we need the game manager called *Parlance*, *negoServer*, *RandomBot* s code and the libraries that it requires, that are *dip* and *bot*.[7]

2. Create a client extending the corresponding classes. Remember that to be able to negotiate you should use *negoClient* and *dipNego* in addition to the libraries specified in Figure 6.9. As this client is a bot, it is not necessary to use *dialogueAssistant* to translate messages to a human.

   We extend *RandomBot* s functionality adding the capability to negotiate randomly thus we also need the language parsing utility, *dipNego*, and the

[6] *RandomBot* is the player that performs random moves introduced in Section 6.4.1.

[7] dip and bot libraries are available at `http://www.dipgame.org/browse/dip`.

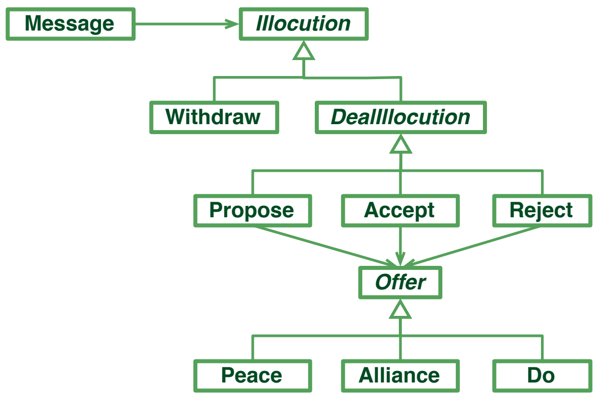


Figure 6.10: UML 2.0 class diagram of the most important classes for L_1 in *dipNego*.

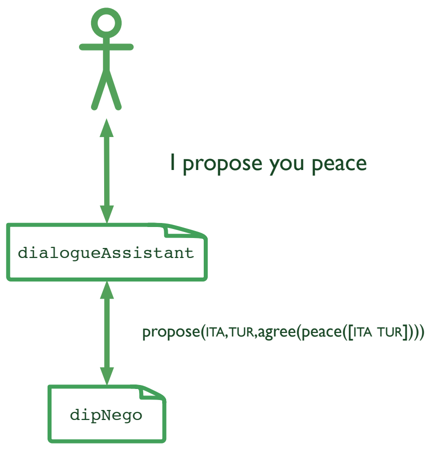


Figure 6.11: Message transformation.

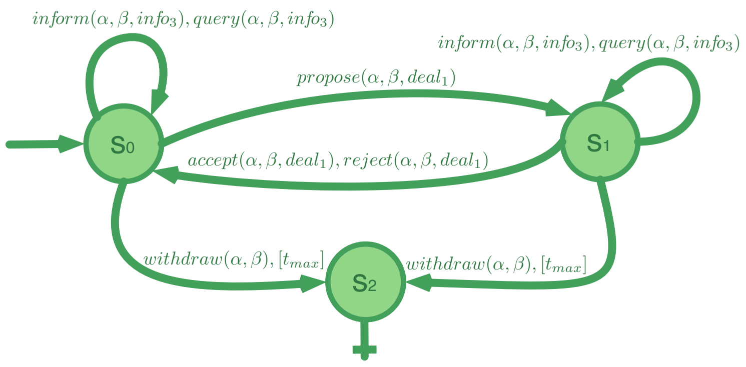

$inform(\alpha, \beta, info_3), query(\alpha, \beta, info_3)$
$inform(\alpha, \beta, info_3), query(\alpha, \beta, info_3)$
$propose(\alpha, \beta, deal_1)$
$accept(\alpha, \beta, deal_1), reject(\alpha, \beta, deal_1)$
$withdraw(\alpha, \beta), [t_{max}]$
$withdraw(\alpha, \beta), [t_{max}]$
S_0
S_1
S_2

Figure 6.12: A communication protocol for L_3.

library that provides the connection with *negoServer* that is *negoClient*. From *negoClient* we must implement the `handleMessage` method indicating what to do when a message is received. Our random negotiator agent would throw a coin to decide whether to accept or reject the received proposal. And if the message is already an accept or a reject, it would do nothing. *negoClient* also allows us to send proposals. Our random negotiator agent would throw a coin at every new turn to decide whether to propose something or not. And the same method would be used to decide whom to propose it and the concrete offer to make, for instance, a peace agreement.

3. Complete your client adding the negotiation model functionality to be tested. This is the step where you integrate your work with the testbed.

   In the example we take decisions randomly therefore the negotiation model simply defines a negotiation strategy based on throwing coins.

4. Set the experiments choosing the language level, the duration of the game and the deadlines in the given servers. Fixed game duration and deadlines for turns are optional but recommended specially when humans take part in the experiments.

   Our example experiment might be composed of games with 7 instances of our random agent. The duration of the game and turns are undefined as this is a simple example where agents make just one proposal per turn before deciding their actions.

5. Run the experiment. This means launching first the game manager and *negoServer* with the desired settings and then running the clients that you want to connect to them.

6. Analyze the results and extract your conclusions. At the end of each game you get the results as a complete log of movements, state updates and exchanged messages.

   From the log files generated we check whether our agent played well and question whether there is a way to improve its performance changing the internal model of the agent.

7. Optionally make the adjustments necessary in your code to improve the performance of your agent.

The code of this random negotiator agent can be found at `http://www.dipgame.org/browse/examples`. More sophisticated agents take advantage of the messages exchanged, the actions performed and the state of the world observed in order to perform smart negotiations.

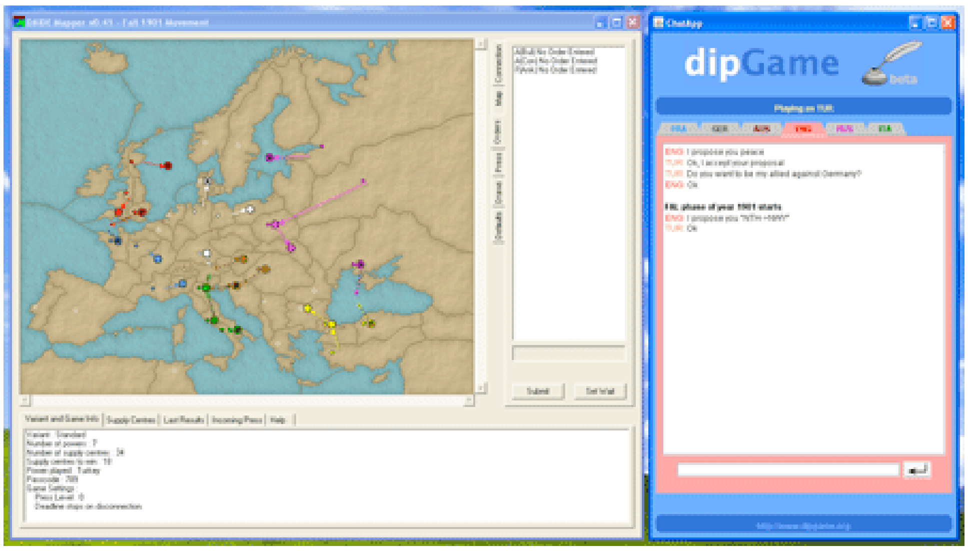
6.5 Running experiments

To simplify the execution of experiments we developed three software tools that we describe in this section. The first tool is a negotiation client for human players to play negotiating. The second is a game manager that runs a DAIDE server, the *negoServer* and collects the data. Finally, we describe *dipTools* that is a software application for analysing the experimental data.

6.5.1 Chat App

The graphical interface for humans to play on DipGame is called ChatApp and can be seen in Figure 6.13. This is a multi-platform interface released as an standalone application. ChatApp provides a chat functionality similar to most instant messaging software available in the market.

The chat translates the natural language messages into the formal language that automated agents understand and vice-versa. This translation is done by the *dialogueAssistant* library in such a way that players do not necessary know whether the opponent is a human or an agent. They must guess that given the behaviour of the players, not the syntax of the messages.



ENG: I propose you peace
TUR: Ok, I accept your proposal
TUR: Do you want to be my allied against Germany?
ENG: Ok

FAL phase of year 1901 starts
ENG: I propose you "NTH →NWY"
TUR: Ok

Variant : Standard
Number of powers : 7
Number of supply centres : 34
Supply centres to win : 18
Power played : Turkey
Passcode : 789
Game Settings
Press Level : 0
Deadline stops on disconnection

Figure 6.13: Screenshot of the human graphical interface, ChatApp. The chat is on the right and the map on the left.

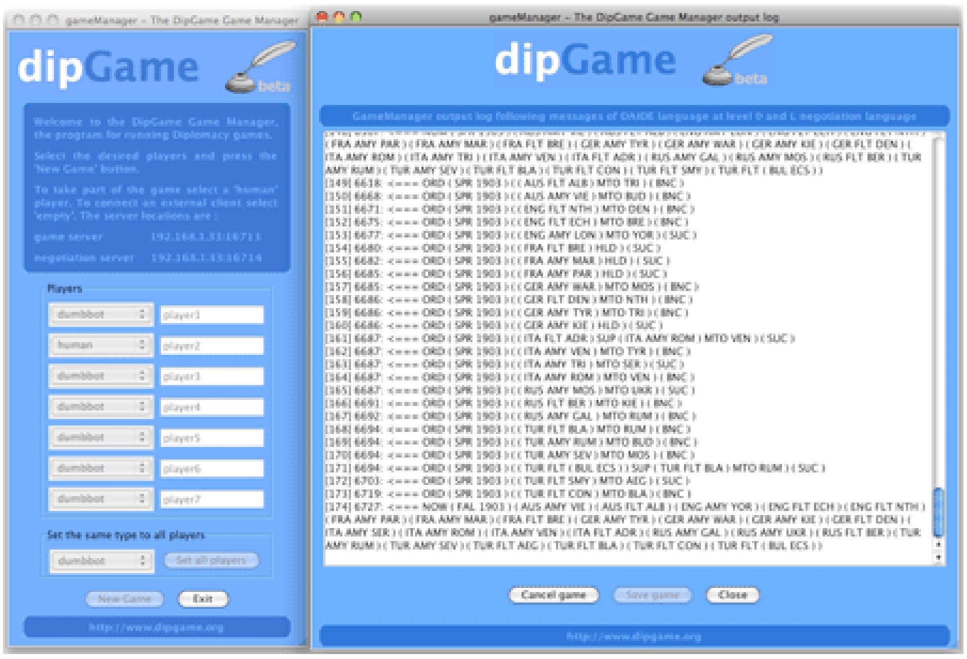
On Windows, ChatApp renders a map used to select the movements to perform. This map is an AiMapper connected to the chat. In other operating systems a ConsolePlayer is used to send the orders.

In order to play, the ChatApp must be connected to a DAIDE server indicating the IP address and port of it and its *negoServer*. Then, if it is supported, it launches the map. Notice that although ChatApp makes use of AiMapper, it controls the mapper and the turns are updated at the same time. Nice graphical and sound effects are included in the application to alert the player about un-read messages and turn update. It also alerts when a message cannot be understood by the *dialogueAssistant* or do not satisfy a requested protocol. Remember that although the DipGame do not constrain the negotiation to follow a particular protocol, if it is necessary, it can be constrained.

6.5.2 Game Manager

The DipGame GameManager, provides a graphical user interface where games can be set up. The application is multi-platform and makes use of AiServer or

Parlance depending on the operating system in use. It launches also negoServer and it controls their execution and collects their results. The application allows you to: (1) select the players you want to take part in the game and (2) run games observing the resulting output log. Thus, each player can be either a provided bot (currently four bots are included in the software release), a bot programmed by the experimenter, or a human interacting through a graphical interface. Figure 6.14 shows an screenshot of the application during a game execution. New bots can be incorporated to the manager being able to be launched by the GameManager. In addition, the manager allows to set a player to *empty*. This means that the game can be launched even though there are players missing. Once launched, the game will wait for those missing players to connect using the IP address and port indicated by the manager. Missing players can be standalone applications running in the same computer for for instance a bot that is not yet incorporated to the manager, or in other computers in the same network. ChatApp is another example of such standalone application.


gameManager – The DipGame Game Manager
dipGame
beta
Players
dumbbot
human
Set the same type to all players
Set all players
New Game
Exit
http://www.dipgame.org
gameManager – The DipGame Game Manager output log
GameManager output log following messages of DAIDE language at level 0 and L negotiation language
Cancel game
Save game
Close

Welcome to the DipGame Game Manager, the program for running Diplomacy games.

Select the desired players and press the 'New Game' button.

To take part of the game select a 'human' player. To connect an external client select 'empty'. The server locations are :

game server 192.168.1.33:16713

negotiation server 192.168.1.33:16714

dumbbot	player1
human	player2
dumbbot	player3
dumbbot	player4
dumbbot	player5
dumbbot	player6
dumbbot	player7

```
( FRA AMY PAR ) ( FRA AMY MAR ) ( FRA FLT BRE ) ( GER AMY TYR ) ( GER AMY WAR ) ( GER AMY KIE ) ( GER FLT DEN ) (
ITA AMY ROM ) ( ITA AMY TRI ) ( ITA AMY VEN ) ( ITA FLT ADR ) ( RUS AMY CAL ) ( RUS AMY MOS ) ( RUS FLT BER ) ( TUR
AMY RUM ) ( TUR AMY SEV ) ( TUR FLT BLA ) ( TUR FLT CON ) ( TUR FLT SMY ) ( TUR FLT ( BUL ECS ) )
[149] 6618: <=== ORD ( SPR 1903 ) ( ( AUS FLT ALB ) MTO TRI ) ( BNC )
[150] 6668: <=== ORD ( SPR 1903 ) ( ( AUS AMY VIE ) MTO BUD ) ( BNC )
[151] 6671: <=== ORD ( SPR 1903 ) ( ( ENG FLT NTH ) MTO DEN ) ( BNC )
[152] 6675: <=== ORD ( SPR 1903 ) ( ( ENG FLT ECH ) MTO BRE ) ( BNC )
[153] 6677: <=== ORD ( SPR 1903 ) ( ( ENG AMY LON ) MTO YOR ) ( SUC )
[154] 6680: <=== ORD ( SPR 1903 ) ( ( FRA FLT BRE ) HLD ) ( SUC )
[155] 6682: <=== ORD ( SPR 1903 ) ( ( FRA AMY MAR ) HLD ) ( SUC )
[156] 6685: <=== ORD ( SPR 1903 ) ( ( FRA AMY PAR ) HLD ) ( SUC )
[157] 6685: <=== ORD ( SPR 1903 ) ( ( GER AMY WAR ) MTO MOS ) ( BNC )
[158] 6686: <=== ORD ( SPR 1903 ) ( ( GER FLT DEN ) MTO NTH ) ( BNC )
[159] 6686: <=== ORD ( SPR 1903 ) ( ( GER AMY TYR ) MTO TRI ) ( BNC )
[160] 6686: <=== ORD ( SPR 1903 ) ( ( GER AMY KIE ) HLD ) ( SUC )
[161] 6687: <=== ORD ( SPR 1903 ) ( ( ITA FLT ADR ) SUP ( ITA AMY ROM ) MTO VEN ) ( SUC )
[162] 6687: <=== ORD ( SPR 1903 ) ( ( ITA AMY VEN ) MTO TYR ) ( BNC )
[163] 6687: <=== ORD ( SPR 1903 ) ( ( ITA AMY TRI ) MTO SER ) ( SUC )
[164] 6687: <=== ORD ( SPR 1903 ) ( ( ITA AMY ROM ) MTO VEN ) ( BNC )
[165] 6687: <=== ORD ( SPR 1903 ) ( ( RUS AMY MOS ) MTO UKR ) ( SUC )
[166] 6691: <=== ORD ( SPR 1903 ) ( ( RUS FLT BER ) MTO KIE ) ( BNC )
[167] 6692: <=== ORD ( SPR 1903 ) ( ( RUS AMY CAL ) MTO RUM ) ( BNC )
[168] 6694: <=== ORD ( SPR 1903 ) ( ( TUR FLT BLA ) MTO RUM ) ( BNC )
[169] 6694: <=== ORD ( SPR 1903 ) ( ( TUR AMY RUM ) MTO BUD ) ( BNC )
[170] 6694: <=== ORD ( SPR 1903 ) ( ( TUR AMY SEV ) MTO MOS ) ( BNC )
[171] 6694: <=== ORD ( SPR 1903 ) ( ( TUR FLT ( BUL ECS ) ) SUP ( TUR FLT BLA ) MTO RUM ) ( SUC )
[172] 6703: <=== ORD ( SPR 1903 ) ( ( TUR FLT SMY ) MTO AEG ) ( SUC )
[173] 6719: <=== ORD ( SPR 1903 ) ( ( TUR FLT CON ) MTO BLA ) ( BNC )
[174] 6727: <=== NOW ( FAL 1903 ) ( AUS AMY VIE ) ( AUS FLT ALB ) ( ENG AMY YOR ) ( ENG FLT ECH ) ( ENG FLT NTH )
( FRA AMY PAR ) ( FRA AMY MAR ) ( FRA FLT BRE ) ( GER AMY TYR ) ( GER AMY WAR ) ( GER AMY KIE ) ( GER FLT DEN ) (
ITA AMY SER ) ( ITA AMY ROM ) ( ITA AMY VEN ) ( ITA FLT ADR ) ( RUS AMY CAL ) ( RUS AMY UKR ) ( RUS FLT BER ) ( TUR
AMY RUM ) ( TUR AMY SEV ) ( TUR FLT AEG ) ( TUR FLT BLA ) ( TUR FLT CON ) ( TUR FLT ( BUL ECS ) )
```

Figure 6.14: Screenshot of the GameManager. On the left side of the figure you can see the main window of the game manager where the game can be set. When running the game, the window on the right side of the figure pops up providing real-time results of the game execution.

When running a game using only a computer, only a human player can play as the screen cannot be shared among several users. When this is the case, the GameManager launches ChatApp and connects it to the game.

Games can be cancelled and the output log stored in a file. The graphical interface facilitates the use of the testbed and provides an attractive and simple way to use the DipGame testbed.

6.5.3 DipTools

The analysis of the data produced by negotiation experiments, consisting of several game executions, can be made with the help of DipTools [Fabregues et al., 2011]. This tool allows the experimenter to group the results of sets of game executions in order to compare and analyse them in a intuitive and graphical way. See Figure 6.15 for an screenshot of the application.

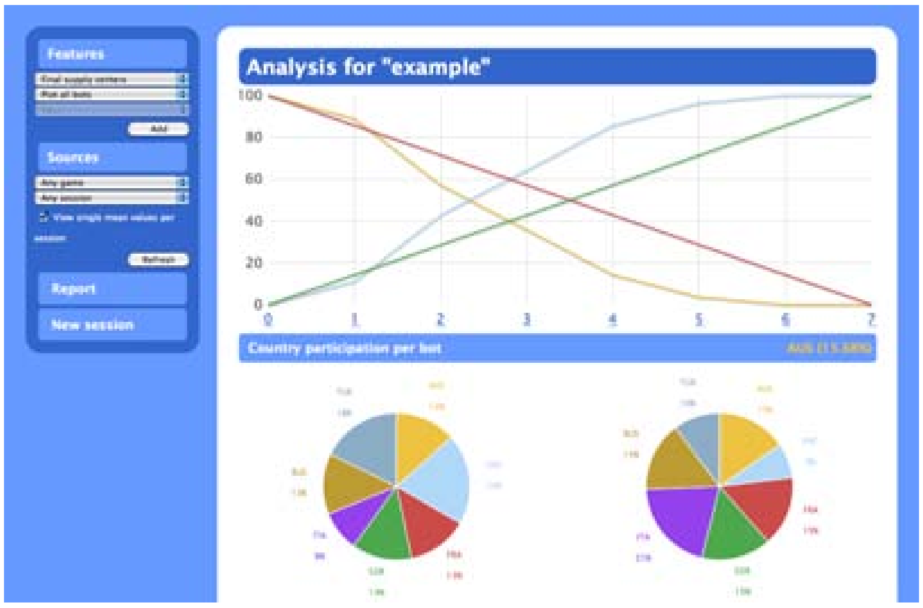


Figure 6.15: Screenshot of DipTools.

An experiment is defined as a set of sessions each one containing a set of games. Sessions are used in DipTools to allow the experimenter to group together the data from games ran using the same settings, it is usually useful to compare results obtained from different settings. Several experiments can be stored but only one can be visualised at any time.

There are three families of charts: (i) for a single game, (ii) for a game session and (iii) for the whole experiment. The chart of a single game represents on the x-axis the phases of the game. On the y-axis it permits to display a numerical variable. For example, the amount of deals reached by an agent.

Given a game session, the tool allows to plot variable values over the games of the sessions. This chart can be used to check whether the performance of a bot was similar or not in all session games. We can plot, for instance, the degree of interaction with other agents or the ranking of the bot at the end of each game.

Finally, given the overall experiment, the tool allows to chart the average of a selected variable over all the games of each session. This option is used to illustrate the results of the experiment described in Section 6.6 and illustrated in Figure 6.17. It is a quick way to visualise the overall performance of our agents.

There are many useful variables that can be displayed and that are related to a player (e.g. the number of successful movements[8]) or to the interaction of two players (e.g. the number of attacks between them). The experimenter just needs to select the observable variables and the involved agents (one or two). An observable variable can be complex as, for instance, the number of times that a given bot has attacked Germany or the number of attacks that it has performed. The tool allows the experimenter to easily define such observable variables, as well as chart several of them at the same time.

In addition to point chart displays, DipTools provides pie charts that are ideal to represent exclusive variable values as, for example, what percentage of victories were obtained by a particular agent depending on what power it was representing. The tool also provides text reports where the data is provided in tabular form.

A video demonstration of the DipTools software is available at `http://www.dipgame.org/media/AAMAS2011demo`.

6.5.4 Methodology

The typical users of the DipGame testbed are researchers that are developing their own software agents (bots). To run an experiment you have to first download the GameManager and incorporate your agent into it. The instructions for incorporating agents are available at Annex A. Next, you can run the manager and set the experiment selecting the players you like to take part in it. Among the available players you will find those software agents that you incorporated. Finally, run the experiment and save the results into a file.

In the previous section we described how to analyse the results of an experiment involving several game executions. To run an experiment involving, for instance, four copies of your bot (each one with possibly different parameter values) and three human agents, you will need at least three machines to interface with the three humans. Three machines would be enough as one might run the GameManager and the other two the ChatApp standalone application. Thus, the GameManager would have four players set to be instances of your bot, one player set to human and the last two players set to *empty*. When running the experiment, the manager launches the game with two players missing and launches also a ChatApp integrated with the manager. This human interface can be used by one of the humans. For the game to be able to start, the other two humans should connect to the manager by introducing in their graphical interface the IP of the gameManager that is shown in its main window. Figure 6.16 illustrates the distribution of software applications per computer and the human players taking part of the game.

The human interface integrated with the GameManager should be used only for testing purposes as the human using it would have access to private messages sent between the other players as the log includes them. When the game ends, or when the game is cancelled,[9] the results can be stored in a file. Then, we can take one or several game result files, load them into DipTools and visualise the results with DipTools.

[8]Sometimes the players do not succeed in performing their movements because of collisions with the movements of other players.

[9]The experiment execution can be canceled at any time from the experiment manager window showing the real time results.

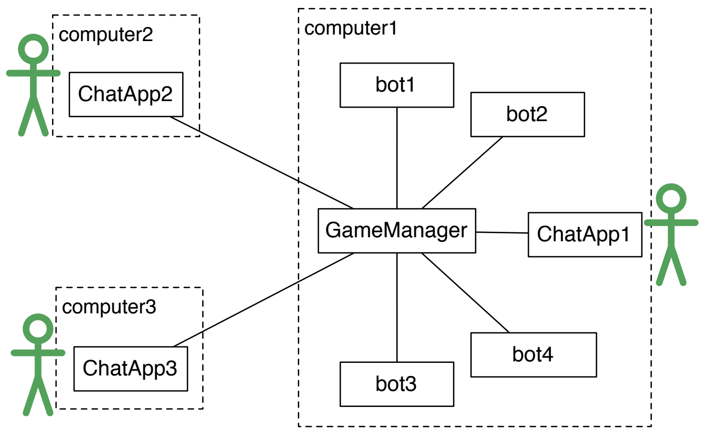


Figure 6.16: Example of distribution of programs among computers.

At `http://www.dipgame.org/media/AAMAS2012demo` you can find a video demonstration of the methodology for running experiments on DipGame. Next section illustrates this with an example.

6.6 Experimentation

To test the correct performance of the testbed we have run many experiments. Most of them were used to check that the functionality included work well. After upgrading the infrastructure many times, fixing bugs and reducing defects, we can now ensure that the infrastructure is completely operative and free of errors. In fact, it has been used by other research groups and students that corroborated this fact. After that, we decided to run an experiment to empirically enforce our claim of being The Diplomacy Game a suitable domain for testing negotiating agents. We prove it running an experiment including several negotiating agents playing against non negotiating agents. In the following we describe the details of the experiment and the results.

6.6.1 Experiment settings

The experiment was set to use the standard version of The Diplomacy Game without convoys, that is used by the testbed. This version is specified in Chapter 4. Negotiation was allowed only during movement seasons and with a duration of 5 minutes. This deadline is necessary to establish the end of the negotiation session between negotiating agents. The game had also a maximum duration of 30 min. Almost all games ended before that time. Those that were still in process were cancelled, and then excluded.

The experiment contained 8 sessions where the games in each session had 0, 1, ..., or 7 instances of a negotiating agent and the rest of the players were instances of a non negotiating agent, e.g. session 4 has 4 instances of the negotiating agent and 3 instances of the non negotiating agent. Both types of agents shared the same game strategy module in order to avoid any possible noise provided by matters that have nothing to do with the negotiation capability of the agent. To obtain a statistically significant result, we executed 100 games per session. The distribution of powers among agents per session was uniform.

6.6.2 Game strategy

The game strategy module of the agents is based on DumbBot, Section 2.5, that has no memory. The inputs of the game strategy are the power that the agent is representing, and the game state, Section 4.3. The output is a complete plan for the power containing all the orders that are recommended to be executed. The game strategy generates the plan assigning an order to each unit. Those orders are selected trying to obtain the maximum possible utility measured as the aggregation of the utility obtained per unit. The utility of a unit is related to the *destination* value of the region were the unit will be at in next turn assuming that the order assigned to it is successfully executed.[10]

The destination value of the regions are computed in several steps. The first step is to assign to each region the initial value given its type and current state. Note that, from the point of view of a power, regions can belong to: general provinces, general supply centres, other powers homelands, or the power homelands. Moreover, given a game state, a region can belong to a: empty province, occupied province, not owned supply centre, supply centre owned by an other power, or supply centre owned by the power. A different initial value corresponds to every single combination of region state and type. Subsequent steps are to iteratively spread those values along the map aggregating for each region a proportional part of the destination values of the neighbouring regions. By this way, destination values show paths towards highly valued regions. The farthest the region, the fewer it is affected. For instance, the highest initial destination value corresponds to a region belonging to one of the homeland supply centres of the power, that it is owned by someone else, and it is not occupied. This is so because owning oneself homelands is one of the basic strategies to play well the game.

The game strategy selects the orders to be performed by each unit trying to maximise the resulting destination value assuming the successful execution of the order. If the highest destination value among the region hosting the unit and its neighbouring regions is the hosting one, then a hold order is assigned to that unit. Otherwise, the strategy assigns a move towards the highest valued neighbouring region. In case of having more than a highest valued neighbouring region, one of them is randomly selected. When all orders are already assigned, hold orders are reviewed checking whether there is an interesting support order for them. This is done using the *competence* and *strength* values.

The strength value is computed as defined in the game specification, Section 4.4.1. The competence value of a province is the maximum number of neighbouring and hosted[11] units belonging to a power that is not the given power. It is a measure of the strongest enemy capable to get the province. Then, hold orders are set to support other orders with less or equal strength than the competence of the destination province. This is also done iteratively until no more hold orders are set to support.

[10]Notice that movements in Diplomacy are concurrently announced often provoking conflicts among them. Those conflicts may end with the cancellation of an order execution.

[11]Note that there can not be more than a hosted unit per province.

The basic distinction between Dumbbot and this game strategy is the distinction between homelands and supply centres, and not owned supply centres and proper provinces. The initial destination values are also adjusted. The use of random variables in this game strategy has been limited to the selection between equally valued regions. Dumbbot use an iterative coin flipping method to avoid a deterministic behaviour. In this experiment, the agents do not model other agents behaviour. Therefore, it is not necessary to avoid a deterministic behaviour. Contrarily, it is beneficial to reduce noise in the experiment results. The number of steps for spreading destination values along the map is reduced to 5 instead of 10.

The game strategy has an optional plan input. In that case, the game strategy outputs a plan completing the input plan with orders assigned to the rest of units controlled by the input power. The process is the same described before. However, the assignation of orders starts with those from the input plan.

6.6.3 Negotiation strategy

The negotiation strategy module generates possible proposals (negotiating options) that improve a given complete plan for a power increasing its utility and reducing the risk of conflicts among orders. The negotiation strategy receives as inputs the power, the game state, and a compete plan for that power. The output is a ranking of negotiating options. Strength and competence values are computed and support orders are assigned to other powers that benefit the input power. No assessment of the other powers utility is done. The agent using this negotiation strategy is assumed to propose the best ranked option. For a received proposal to be accepted, it must provide a plan as good or better than the plan initially proposed by the game strategy. To generate the best possible plan including the received proposal, the negotiation strategy uses the game strategy with the optional input plan set to the received proposal. Then plans are compared based on their utility and risk of conflicts that are measured as described before. If the received proposal is not good enough, then the proposal is rejected.

6.6.4 Execution

Both agents are developed using the *dip* framework and the game strategy module. In addition, the negotiating agent uses *nego* and the negotiation strategy module. We can execute a game in a machine using the GameManager. To that end, we need to incorporate our two new agents and then start the GameManager. Some players will be set to the negotiating agent and some others to the non negotiating agent. Then, the game can be executed. Notice that when we execute a game on a single computer, we have more processes to execute than processors available, and the process dispatcher is controlled by the operating system. This is usually the case as there is a process per player and current development computers use to have only one or two processors. In this setting, we cannot be sure of all processes (players) being in equal opportunities to negotiate. Moreover, the execution of 800 games in such computer would take 17 days. We decided to run the experiment in a supercomputing centre.

The execution of the experiment has been done in the super-computing facilities of the Barcelona Computing Center (`http://www.bsc.es`). Concretely, we used Marenostrum that is a machine with 2554 JS21 blades PowerPC 970MP of 4 Cores 8 GBytes memory that provide a final calculation capacity of 94.21 Teraflops. Every blade is a node that allow the concurrent execution of 4 processes sharing the same memory. Access to the machine was subject to disposability. Execution tasks are defined as jobs and are queued to be executed when decided by the job dispatching system.

The execution of the experiment require 9 processes: one per player, one for the game server, and another for the negotiation server. All of them use several threads for communication and computation. For instance, a non negotiating agent uses a thread for listening the game server and another for the general agent execution that generates the orders to perform. A negotiating agent has, in addition, a thread for listening the negotiation server and another for negotiating. We require a node per player in order to have enough cores to allocate all their threads. For the servers, we use a node only because they are not needed at the same time. Negotiation server operates while the game server is waiting for the phase deadline. This distribution of processes per nodes guaranties the same communication speed for all of them as the Marenostrum assigned nodes connected to the same switch.

The specification of the job was specially difficult to set because all processes need to run at the same time, use the 4 cores of a node, and know the location of the servers node. Players need to connect to the server node and it was initially unknown. The specific node to allocate a process is decided by the job dispatcher. To enable access to the servers node address, we used an special feature of the SLURM resource manager that launch multiple programs at the same time. This allowed us to specify what process must run in what ordinal node from the list of nodes provided at loading time by the dispatcher. For instance, the servers run in the first node of the list. Then, the players must refer to the first node address as the servers address.

A job was defined per session involving the sequential execution of games. Therefore, a job take a maximum of 2 executing days. And all jobs could be executed at the same time. The log files generated by players and servers was stored in a temporal high speed memory and moved to a disk at the end of each game. These log files contain the results of the experiment. Notice that the game server to use is Parlance as it can run on a unix system.

6.6.5 Results

After running the experiment, 800 games, we load the log files containing the results of the experiment into DipTools. We choose *any session* and *any game* as source data in order to represent sessions at the X axis. We choose *winner* and *all bots* as variables, and mark the single mean checkbox to represent the percentage of games won by bot. The resulting chart is illustrated in Figure 6.17 were the percentage of games won by every agent is represented per session.

This experiment is set to have a different number of players of each agent type per session. Session 0 corresponds to the session with games containing 0 negotiating agents. Session 1 contains 1 negotiating agent, and correspondingly the rest of sessions. We can say that the negotiating agent performs better than the non negotiating one because its percentage of victories is larger than the expected results in case all agents were of the same type.

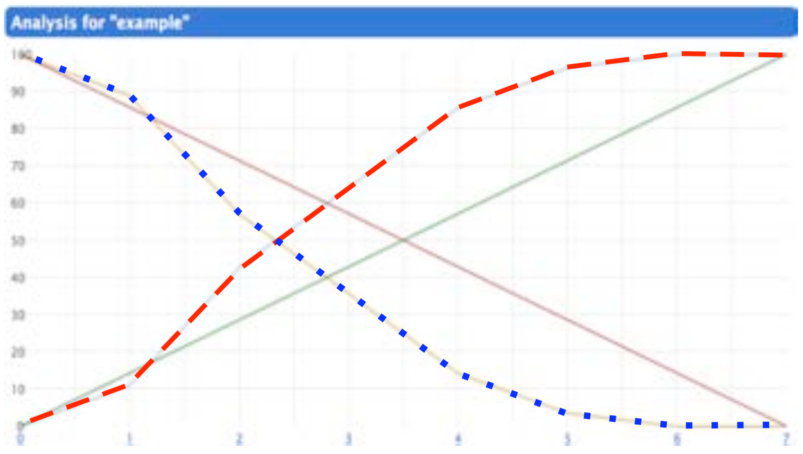


Figure 6.17: Chart of the experiment. Percentage of games won per number of negotiating agents. The dashed line represents the percentage of victories of negotiating agents and the doted line the percentage of victories of non negotiating agents. The continuous lines (increasing and decreasing) represent the expected percentage of negotiating and non-negotiating agents in case they all were equal. This particular graphic shows that negotiating agents perform better in the experiment than what is probabilistically expected.

6.7 Summary

In the last decades, several software developers have build Diplomacy players. Most of them use DAIDE and rely on its communication protocol and language. DAIDE establishes a client-server protocol where all Diplomacy players are clients and the server organises the communication and contains the adjudicator algorithm. DipGame can be seen as the adaptation of DAIDE to the MAS research community. DipGame reuses as much content as possible from DAIDE and provides extra resources desirable for MAS researchers. One of the differences between DAIDE and DipGame is the negotiation language used in them. DipGame s negotiation language is denoted by L and it is a language hierarchy separating the Diplomacy vocabulary from the domain independent terms and the grammar. This is made operative by the use of a negotiation server connected to the DAIDE server and by forbiding the use of DAIDE server for negotiation purposes. The DipGame negotiation server is denoted by negoServer.

Several bot development frameworks and communication APIs exist around DAIDE. Most of them are dependent of a particular operating system, Windows. This dependence complicates the evaluation of agents because most of the computing centers use other operating systems. We decided to build our own bot development framework in a multiplatform fashion making use of an already existing Java API for Diplomacy, the so called *jac* library. This framework is denoted by *dip* and provides an object oriented representation of the game and a player scheleton that simplifies the development of a player a lot. Complementing the framework, the *nego* package provides support for negotiation connecting to *negoServer*, and *dipNego* provides an object representation of L that facilitates the generation of sentences. The same *dip* framework can be used to create other DAIDE clients as, for example, an interface for human players to take part in the games. We have developed a library to translate from (a restricted set of) English into L. This library is denoted by *dialogue-Assistant*. By default, the negotiation protocol is completely free. However, it is possible to constrain communication forcing, for example, proposals to be replied or communication to be bilateral.

Developing a player using the above mentioned framework and libraries consist in extending a class completing it with the desired agent decision model. Running experiments consists in executing several games with particular combinations of players and values for the bots. To facilitate the initial evaluation of agents, we have created *chatApp* that is a DAIDE client with a mapper and a chat that makes use of *negoClient* and, thus, connects to the DipGame *negoServer*. This software has a nice graphical interphase and makes use of *dialogueAssistance* to provide human like sentences to the users. In addition, we created the *GameManager* that organize the execution of games running a DAIDE server and a negoServer with the game settings decided by the user that use a nice graphical interface to set those options. This program provides a complete log of the game and the negotiations that take part in it. The content of this log can be exported for future analysis of results. Finally, DipTools is the program that we provide for analysing those results grouping similar games and analysing their results comparatively providing tables and charts. This section included the description of an experiment using a supercomputing facility and this tool. The experiment proves that The Diplomacy Game is a suitable domain for testing negotiation and that, therefore, DipGame testbed is a suitable one for testing negotiating agents.

Chapter 7

Application to the game industry

The selection of a game as the domain for experimentation offers the possibility to apply our work to the game industry building software agents capable to play. We have applied our work to the online game industry providing a solution for playing Diplomacy online. Moreover, we have integrated our agents into Droidippy, a mobile solution for playing Diplomacy. To disseminate the DipGame testbed, we have created a website with all documentation and resources of the testbed. This website is also the host of our solution for playing Diplomacy.

In this chapter, we start motivating the application of the work to The Diplomacy Game in Section 7.1, and proceed describing the DipGame website in Section 7.2. The description includes the player section (Section 7.2.1) with the online game industry solution, the researcher section (Section 7.2.2) with the testbed documentation and resources, and an analysis of the impact of the website since its publication (Section 7.2.3). Then, we describe the Droidippy integration of DipGame in Section 7.3. The chapter ends with a summary in Section 7.4.

7.1 Motivation

Diplomacy is a very popular game but its duration, up to 2 hours, makes quite difficult for a player physically to congregate another 6 people to play a game. Even when that is possible, face-to-face[1] games complicate the negotiation part of the game as players need to take turns to negotiate, and others can see who is meeting with whom. An additional problem is that players know who are the rest, and could use knowledge from previous played games or take personal relationships into consideration. E.g. Someone could be more motivated to defeat a brother than a daughter. All those features related only to face-to-face games can be seen as part of the game and accepted. Contrarily, most of the players seem to prefer keeping a hidden identity, and being able to meet with

[1]A *face-to-face* game is a game where all players are physically in the same place using a commercial board game edition and sitting around the board. Private negotiations can take place by turns in a different room.

a player without being seen by the rest. All those players use to play a postal, e-mail, or online version of the game. We assume that nowadays most of them play online. Playing online facilitates the congregation of users hiding their identity until the end, and keeping conspiracies undercover.

7.2 DipGame website

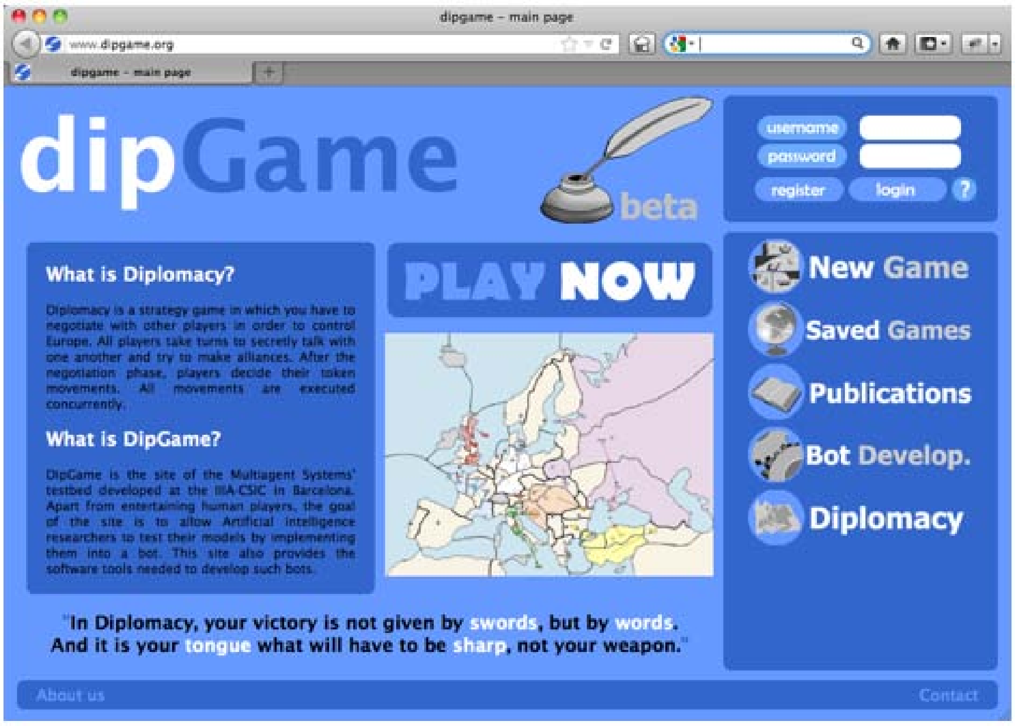
The DipGame website is the site of the DipGame testbed and contains all documentation and the software resources related to the testbed. In addition, it provides access to a web application for playing Diplomacy games. The users of the site are either players of Diplomacy or researchers seeking for information to build agents that can be tested with the DipGame testbed. Those software agents are bots that can play against human players from all around the world through the DipGame website. The users are informed that, by playing, they are participating in an experiment. This is done, at the very beginning, with a description of DipGame at the main page of the site. Users are aware that some of the adversaries will be bots and that all data is stored for research purposes.

Many of the new users know the game, are supporters of it and come to DipGame looking for some training. They are aware of the difficulties of the project and know the alternatives for playing the game. Other players are newbies that are used to play other games online. We understand that we cannot ignore those potential newbie users and must provide an attractive user interface to keep them around. A short description of the Diplomacy game is included in the main page of the site with a representation of the Diplomacy map, and there is a section about Diplomacy accessible from the same main page.

The other class of users is the researchers. They use DipGame at work and, although they thank from having nice interfaces; they expect something serious. The graphical design of the website intends to satisfy everybody using funny but pastel colors, round corners, simple texts messages and slogans, and round and very intuitive icons.

The Diplomacy Game situates the player in Europe at the beginning of the 20th century. Those are the years previous to the Cold War. Despite the fact that chips represent military units and that the vocabulary is quite warlike; we understand that this is a game that is more about negotiations than battles. We reflect this view in the site by the use of light colors, an ink pot as the site s logo, and the following slogan: In Diplomacy, your victory is not given by swords, but by words. And it is your tongue what will have to be sharp, not your weapon. . Figure 7.1 shows a screenshot of the main page of the website containing all the previously mentioned graphical features. At the right side of the image, we have at top the login form and access to the registration form. Bellow it, the main menu panel with the buttons giving access to the main sections of the site. Some of the sections require users to be logged in, concretely the first two, the rest do not. In the following Section 7.2.1, we describe the website s main sections for players, and in Section 7.2.2 those for researchers.

The last main section in the menu describes The Diplomacy Game providing a video where we illustrate a particular game using 3D effects. The video contains an intuitive narrative of a specific game scenario explaining the decisions made by players, their expectations, and their emotions during that short part



What is Diplomacy?

Diplomacy is a strategy game in which you have to negotiate with other players in order to control Europe. All players take turns to secretly talk with one another and try to make alliances. After the negotiation phase, players decide their token movements. All movements are executed concurrently.

What is DipGame?

DipGame is the site of the Multiagent Systems' testbed developed at the IIIA-CSIC in Barcelona. Apart from entertaining human players, the goal of the site is to allow Artificial Intelligence researchers to test their models by implementing them into a bot. This site also provides the software tools needed to develop such bots.

"In Diplomacy, your victory is not given by swords, but by words. And it is your tongue what will have to be sharp, not your weapon."

Figure 7.1: Screenshot of the DipGame website s main page.

of a game. At the bottom of the main page, we include contact info and access to a page listing the people and research projects that support the website.

7.2.1 Player section

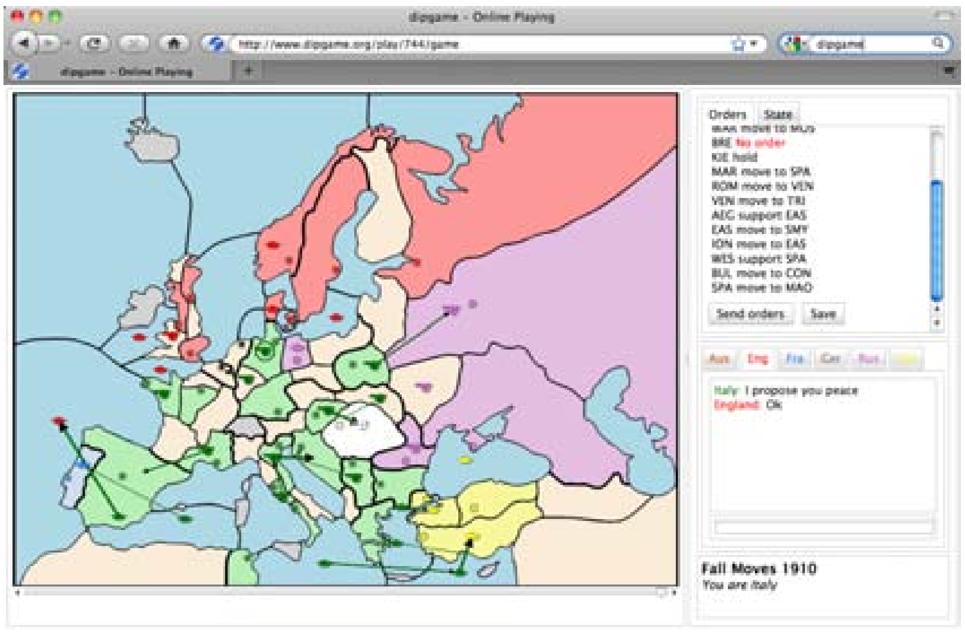
The sections specially designed for players are personalised. Users are required to login in order to visit them. Non logged users are redirected to a different page to login or register. Registration requires only a valid email account, a user name and a password. This authentication allows us to identify players letting them login using the user name and password. The email address is validated through a confirmation mail, and it is used to change the password in case of being it forgotten by the user. Several security techniques are used like password encryption. Those sections for players include the web application for playing games and a page that lists the ongoing and finished games of the user.

The web application provides an interface similar to ChatApp, see Section 6.5.1. The games are managed by a Parlance server[2] connecting to it bots and humans using the DipGame web interface. The web interface is composed of an interactive map representing the state of the game: unit positions and types, and supply centre ownership. The map is the main part of the interface and allows players to indicate their orders clicking on the units and moving the

[2]Parlance is a DAIDE server available at `http://pypi.python.org/pypi/Parlance/1.4.1`.

mouse over it. The map is the standard version of jDip s SVG map[3] thus, the correct visualization of the map depends on whether the web browser can render SVG images or not.[4]

The Figure 7.2 shows a screenshot of the application representing an ongoing



MAR move to MUS
BRE No order
KIE hold
MAR move to SPA
ROM move to VEN
VEN move to TRI
AEG support EAS
EAS move to SMY
ION move to EAS
WES support SPA
BUL move to CON
SPA move to MAO

Italy: I propose you peace
England: Ok

Fall Moves 1910
You are Italy

Figure 7.2: Screenshot of the dipgame web application for playing Diplomacy.

game where Italy is about to win. To the right side of the screen, we can see the message instant tool (the chat). The interface is similar to the chatApp, but in this case, the chat is embedded into the same page containing the map. Below the chat, a status panel shows the year and season denoting the current turn of the game. The original terminology is used for seasons meaning that *Summer* and *Autumn* are denoted by *Spring moves* and *Fall moves*, and *Winter* is now *Adjudication*.[5] This panel also informs about the power that the player is incarnating in the game. In addition, some help on how to indicate orders is provided by the representation of the name of the province that the mouse is currently hovering on, and a message indicating whether the order that the user is indicating is writable or not. For instance, in Diplomacy armies are not allowed to be over sea provinces. Thus, if the user tries to move an army into the sea, the move is not set and a message reporting the problem is shown in the status panel.

Above the chat there, is a panel with two tabs: Orders and State. The Orders tab contains a list of all written orders and buttons for sending them, save the game and cancel the game. By default, all games are saved. Initially,

[3]jDip is an adjudicator and mapper available at `http://jdip.sourceforge.net/`.

[4]The most popular web browsers are able to render SVG images with the exception of Explorer that needs some additional software to do that.

[5]Section 4.1 describes the terminology used for Diplomacy seasons in DAIDE and in the original rules of the game.

the list of orders contains only the power s units with no order assigned to them. By this way, we remember to the user that he/she must decide an order for each of those units. The State tab contains a report of the current state of the game including the number of supply centres owned by power. It also contains two checkboxes enabling the printing of province names on the map. Short and complete names can be represented.

The chat allows the player to negotiate with the rest of players using restricted natural language. As introduced in Chapter 6, we provide a parser for restricted natural language. It is called *dialogueAssistant* and it translates written sentences into an equivalent sentence in L. When the interpreter is not sure about the meaning of those inserted sentences, it displays a message to the user requesting him/her to rephrase them. *dialogueAssistant* is available for messages at level L_1.

When orders are sent, and the new turn begins, the orders decided by all the players are represented on the map indicating which of them were feasible and the new positions of the units. If there is any, the dislodged units are also represented. Players with no dislodged units go directly to the next turn with no interaction required. For the players to be able to check previous turns, a slider situated below the map can be used to set the desired turn. Then, the turn is represented on the map.

As introduced before, there is a page where the logged users can see the list of their games including those that did already end. Those ended games can also be observed and studied being represented in the same interface but with the chat and other buttons disabled. In that case, the slider is crucial to browse through the turns.

DipGame is very attractive for former Diplomacy players and also for newbies. The Diplomacy Game is not difficult to play in DipGame. The individual plays are quite simple, and the interface is very friendly to use and intuitive. It provides help facilities for newcomers to familiarise with the game. In our experience, it is a very attractive game and people get addicted to it quickly. Players even wait to observe the game after being eliminated, and they feel the need to express their opinion on the game at the end. This is very important because it makes easy to motivate them and it lets us get extra information in a poll.

The motivation is another common problem. Current negotiation testbeds (see Section 2.3) define artificial environments where humans have to learn how to behave. Usually, there is no simple story behind the task/game and the actions/movements to perform. The negotiation between players uses to be unreal and very restricted. In our case, Diplomacy is broadly known, and it is also very easy to learn. The argument of the game does quickly involve the players and keeps them motivated. Experiments in the economic field use money to motivate the human participants, but we think that the game is motivating enough, and the honour of winning it cannot be compared to earning money. Even though, a monetary award can be offered to the winner if desired.

7.2.2 Researcher section

The sections designed for researchers do not require users to be registered. Those sections include a page that lists the publications related to the DipGame testbed and the bot development section. This section contains a description of the testbed and gives access to several pages where the testbed components are described. Those pages are called: dip, nego, gameManager and chatApp. All documentation and the software resources of those components are included in these pages. This section does also include the tutorial incorporated to this thesis as Annex A. A video demonstration was created to disseminate the testbed on the multiagent systems community. It is included in this section. The video introduces the game, motivates its use in research and describes the testbed.

Bots taking part of the online Diplomacy games must be previously tested off-line with the GameManager. This is crucial to ensure that software agents perform well and do not crash while playing a game in the DipGame website. Notice that execution errors demotivate the users. We desire human players to feel comfortable using our site because we want them to continue taking part of experiments. Keeping them motivated is crucial.

7.2.3 Impact

The DipGame website is in production as a beta version at `http://www.dipgame.org` since December 2009. It gives access to humans to play Diplomacy on-line against some of our bots. Although we did not make any advertisement to the Diplomacy players community, the website gets requests to play from many users. We have a total of 344 registered users in June 2012, and there are several groups of researchers developing bots over the platform. Also, the testbed is being used for academic purposes in bachelor degrees and master degrees [de Jonge, 2010; Polberg et al., 2011] (e.g. `http://www.cse.unr.edu/robotics/bekris/cs483_s10/handouts`).

Before September 2010, we store the registration date of new DipGame website users. Until then, the site had 103 successful registrations.[6] Figure 7.3 illustrates the rest of the posterior successful registrations in a histogram repre-

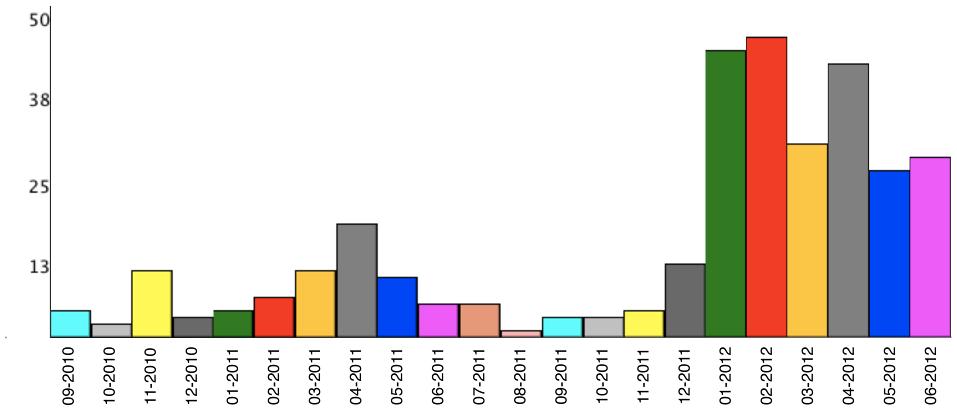


Figure 7.3: Number of registered users per month. Note that, before September 2010, we had a total amount of 103 registered users.

senting the number of registered users per month. Between September 2010 and

[6]Testing users and non confirmed registrations are not included in the statistics. All registered users confirmed the registration mail sent from the site.

the end of 2011, we had in average one or two users registered per week. 2012 supposed a huge increment of registrations now having more than a registration per day. We assume that this increment is related to the inclusion of DipGame in the Wikipedia s page about The Diplomacy Game[7]. Since then, the DipGame website was advertised only among the multiagent systems research community giving talks at the following conferences and workshops: Autonomous Agent and MultiAgent Systems (AAMAS) [Fabregues et al., 2010, 2011; Angela Fabregues, 2012], EUropean MultiAgent Systems (EUMAS) [Fabregues and Sierra, 2010] and Agreement Technologies COST ACTION workshops. The users of DipGame website before 2012 were researchers or friends of researchers. Since 2012, some Diplomacy supporters are also part of the DipGame website.

Figure 7.3 shows the number of unique visits[8] per month since May 2010.

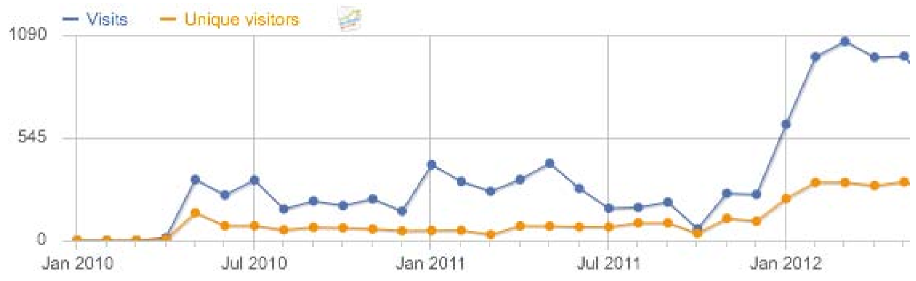


Figure 7.4: Unique visits per month. The tracking system was enabled in May 2010, there are no previous data. The October 2011 local minimum corresponds to a period of time with the tracking system disabled.

The unique visitor results go in line with the number of registered users per month. This data are recorded by duplicate using two web analytical tools: StatCounter and Piwik.[9]

7.3 Droidippy integration

Droidippy is the last solution appeared for playing Diplomacy. It provides the same features than other solutions and an additional one that is very interesting from our point of view. The main difference between DipGame and the rest is the possibility to play against bots. In the case of Droidippy, what makes it different from the other solutions is that it is possible to play from a mobile device using a dedicated mobile application. Droidippy provides a web and an Android[10] application letting games include players that play from a computer or a mobile phone indistinctly. We appreciate that feature and contacted the Oort Cloud company[11] that is powering this system. They accepted to collabo-

[7]The Wikipedia's page about Diplomacy: `http://en.wikipedia.org/wiki/Diplomacy_(game)`.

[8]A unique visitor is counted for all pages loaded in a given session.

[9]StatCounter and Piwik are web analytics software available at `http://www.statcounter.com` and `http://piwik.org` respectively. We installed piwik in DipGame's server and use StatCounter on-line.

[10]Android is one of the most popular operating system for mobile device.

[11]The website of the Oort Cloud company is at `http://www.oort.se`.

rate with us allowing DipGame bots to take part in Droidippy games obtaining from Droidippy all the data related to those games. This means that more users would be able to take part of our experiments and we save the cost of developing a mobile application having only to develop the integration of DipGame bots into Droidippy. Next Section 7.3.1 describes Droidippy and Section 7.3.2 describes the integration of the systems.

7.3.1 Droidippy

Droidippy is an online solution to play Diplomacy that allows users to play long games involving humans only. The games can be set with different time deadlines for turns. The default and most popular setting is a turn per day. Games, thus, take several weeks, maybe months. The dynamics of the games are similar to those played by e-mail. Negotiating messages are in English. In general, the game proceeds generating the new turn when all players have submitted their orders. If the deadline is reached before that, the turn is also resolved with not-yet-ordered units set to hold. The norms of the system impose that when the missing orders are from powers that did not submit in time in the previous turn, then the turn is resolved anyway. This norm let games proceed when someone is missing.

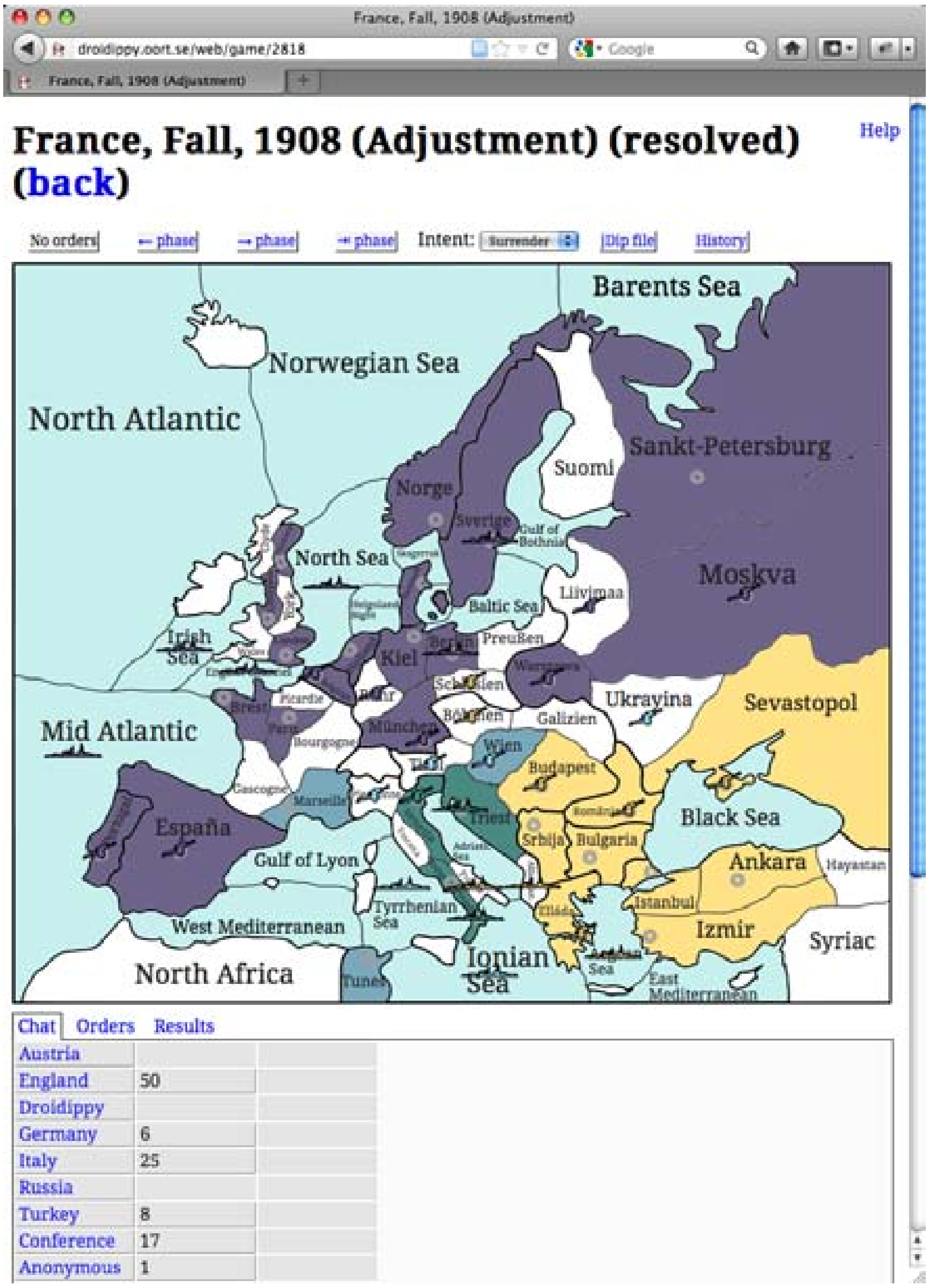
During the progress of a game, players can connect to Droidippy either using the web application or the mobile application. Players can connect several times during a turn and with different applications. They can even negotiate with one of the applications and submit the orders with the other. Several games can be played per player at the same time.

Droidippy does not use DAIDE. It uses jDip for the mapper and also for the adjudication. jDip is an open source Java application that provides a graphical user interface with a mapper. It allows to study previous played games, to check adjudication rules, and it can also adjudicate face-to-face games facilitating the work of master players. Droidippy uses its own game manager that communicates with the web and it mobile applications, and resolves turns with jDip s adjudicator.

Web application

Droidippy s web application allows users to play Diplomacy games. It has similar features to DipGame but with different graphical design, different chat representation, and they connect to different game managers. Its map is also the jDip s SVG map. Thus, as DipGame, Droidippy s web application depends on whether the web browser can render SVG images or not. The web application is in production at `http://droidippy.oort.se`.

In Figure 7.5 , we can see a screenshot of the web application s map page. The slider used by the DipGame web to allow users to check previous turns is replaced here by the `←*phase* link situated above the map. Every order introduced is automatically saved. Despite of that, it is necessary to submit to indicate that orders have already been selected, and that, therefore, the user is waiting for the next turn. The representation of the chat is quite different to DipGame s one. Instead of using a tab per power, powers are listed, and a new panel is shown when we select a power. In that new panel, the representation of previous messages is the usual with new messages in the bottom and reporting

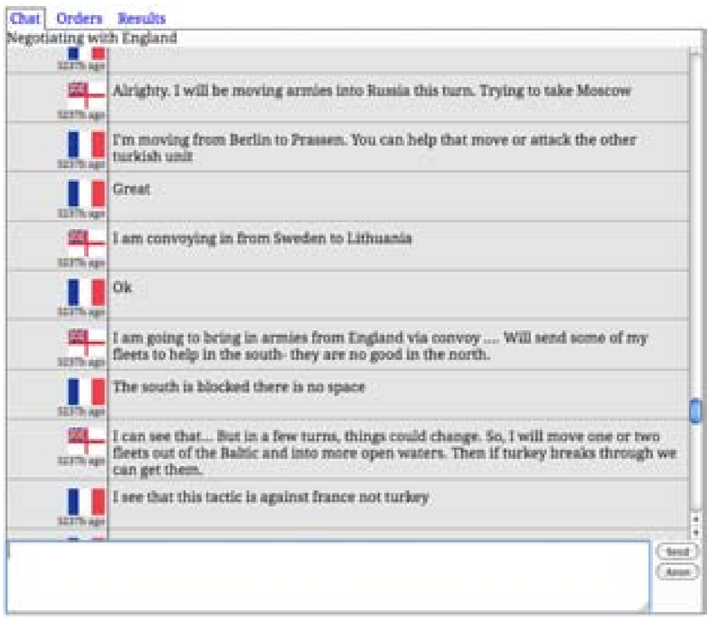


France, Fall, 1908 (Adjustment) (resolved) (back)

Austria		
England	50	
Droidippy		
Germany	6	
Italy	25	
Russia		
Turkey	8	
Conference	17	
Anonymous	1	

Figure 7.5: Droidippy web application s map page with power list.

new turns and message sending times. In Figure 7.6, the negotiation panel is



Alrighty. I will be moving armies into Russia this turn. Trying to take Moscow

I'm moving from Berlin to Prassen. You can help that move or attack the other turkish unit

Great

I am convoying in from Sweden to Lithuania

Ok

I am going to bring in armies from England via convoy .... Will send some of my fleets to help in the south- they are no good in the north.

The south is blocked there is no space

I can see that... But in a few turns, things could change. So, I will move one or two fleets out of the Baltic and into more open waters. Then if turkey breaks through we can get them.

I see that this tactic is against france not turkey

Figure 7.6: Droidippy web application s negotiation panel.

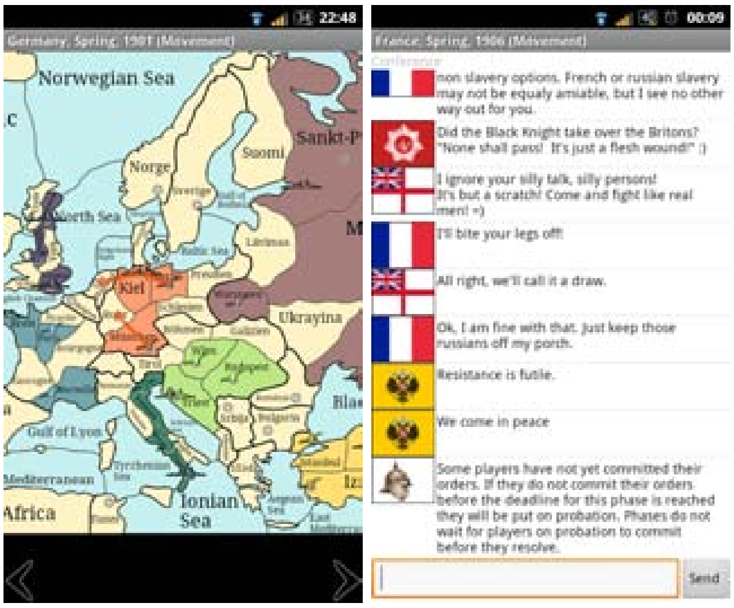
shown.

Mobile application

The Droidippy s mobile application allows users to play Diplomacy games from mobile devices. It is similar to the web application but allows to set alerts and event notifications. It is compatible with Android 2.2 onwards. The first time that it is being used, it needs to be configured for a given user. Then, the first panel that is shown is a list of the user s ongoing games. The user can select a game, or go to the menu to join or create a new game. When playing a game, three panels are enabled: the map, the power list and the orders. The map page is illustrated in Figure 7.7 (left). The interaction with the map is similar to the web application one, and powers are listed in the similarly. Two arrows situated in the bottom of the panels allow the user to change the view from a panel to another. When selecting a power from the power list, the typical chat view is shown with the list of previous messages historically organised. A screenshot is included in Figure 7.7 (right). The orders panel contains all orders that were indicated by the user on the map. The current turn orders are automatically saved.

7.3.2 Integration

To integrate DipGame bots into Droidippy, we just need to use a new communication library. All software from the DipGame testbed uses the DipGame bot development framework (*dip*) to communicate with the servers using DAIDE s



non slavery options. French or russian slavery may not be equaly amiable, but I see no other way out for you.

Did the Black Knight take over the Britons? "None shall pass! It's just a flesh wound!" :)

I ignore your silly talk, silly persons! It's but a scratch! Come and fight like real men! =)

I'll bite your legs off!

All right, we'll call it a draw.

Ok, I am fine with that. Just keep those russians off my porch.

Resistance is futile.

We come in peace

Some players have not yet committed their orders. If they do not commit their orders before the deadline for this phase is reached they will be put on probation. Phases do not wait for players on probation to commit before they resolve.

Figure 7.7: Screenshots of the Droidippy application.

protocol. As explained in Section 6.4.1, *dip* has a separated module called *daideComm* to deal with DAIDE s communication protocol. This module uses the Java Communication API (*jac*) to encode messages, and it connects to DAIDE servers by TCP/IP. To communicate with Droidippy s manager, instead of communicating with DAIDE servers, we need to use a new module called *droidippyComm* instead of *daideComm*.

The *droidippyComm* module connects to Droidippy s manager by HTTP requests. The manager can respond to those requests, but it cannot request anything to the clients. In fact, if TCP/IP requires client and server to be connected, HTTP establishes a connection every time it requests for information. Therefore, clients must keep periodically requesting for new info. *droidippyComm* does it, and parses all received info in order to represent it in an object oriented structure compatible with DipGame. To negotiate, DipGame clients use *negoClient* that connects to a *negoServer*. There is no DipGame s *negoServer* in Droidippy s manager. It has its own embedded negotiation server. Therefore, *droidippyComm* must act as a *negoClient* and send those messages to Droidippy s manager instead to a *negoServer*.

All the above described functionality is already implemented. They still did not release in production the play-against-bot feature that allow their users to play against our bots. They released it in their pre-production site where we have been testing it. Despite of that, the integration is done, and it is operative in pre-production where it has been fully tested. Figure 7.8 represents the module structure necessary for DAIDE and Droidippy to be used.

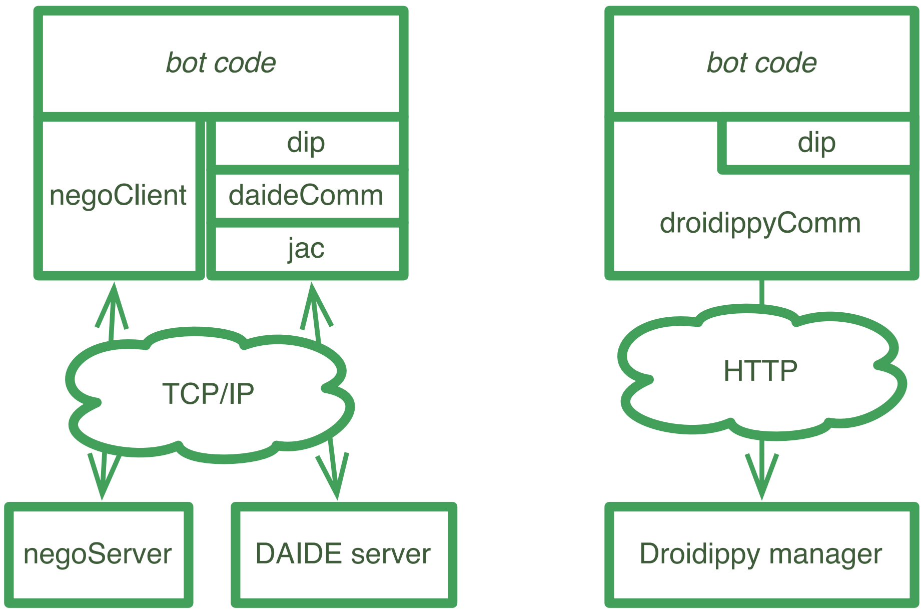


Figure 7.8: Bot module structure for DAIDE and Droidippy.

7.4 Summary

The website dedicated to the DipGame testbed is available at `http://www.dipgame.org`. We have created this website to publish in it all the testbed documentation and resources, including related publications and video demonstrations. Moreover, the DipGame website provides access to a web application that we have created with the aim of applying our work to the online game industry. This web application is similar to other solutions for playing Diplomacy online: it contains a map, a chat, several games can be played by player at the same time, ... What distinguishes this solution from others is that it allows to play against software agents (bots). We have designed the web interface taking into account the diversity of potential players that could use it. Instead of focusing our efforts on congregating Diplomacy experts, we tried to build an attractive solution for all audiences, including newbie players. All those players take part of our experiments because we use the data obtained from games to analyse the performance of the software agents and improve them. The impact of the website is considerably high (1000 visits per month) taking into account that we did not advertise the website in any Diplomacy player community. The work was disseminated only as a testbed in research conferences and journals.

Droidippy, available at `http://droidippy.oort.se`, is the most recent solution for playing Diplomacy online. It differentiates from DipGame and others because it allows to play from a mobile devide. We have collaborated with Oort Cloud, the mobile game company powering Droidippy, integrating our software agents in their system. By this way, all software agents developed following DipGame testbed can play Droidippy games. This collaboration benefits Oort Cloud providing a new feature for their system: the feature of being able to play against software players. It also benefits DipGame providing more human players for the experiments, and all the data related to those games. The integration of DipGame into Droidippy is fully tested using Droidippy s pre-production site.

Chapter 8

Discussion

In this chapter, we discuss the thesis analysing the contributions, and comparing them with the background. Then, we mention some future research work, and we conclude listing our publications that are related to the work described in this thesis.

8.1 Contributions of this thesis

In this thesis, we face the challenge of automated negotiation with humans focusing our attention in several aspects that were not studied in depth in the past, like pre-negotiation and multiple bilateral negotiation. We define a negotiation problem, RNP, with several potential applications that may benefit humans in their everyday life. The use of a negotiation protocol allowing multiple bilateral negotiations is crucial to that end, as they are common in business [Bichler et al., 2003]. Also the fact that negotiations in this problem are about actions, and repeat along time facilitating the establishment and evolution of relationships among the agents. Deadlines are assumed for negotiations as they take place before actions are performed.

An architecture for agents to take part in RNP is defined. HANA follows a heuristic approach and deals with pre-negotiation. The architecture defines how observed information must be represented. It solves the problem of supplying the negotiation strategy with plans and negotiating options assuming a huge space of possible solutions. To that end, it uses an anytime algorithm that runs in parallel to the negotiation strategy providing an up to date ranking of plans and options. The architecture takes into account the cost of exploring the space of possible solutions. Several negotiation strategies are proposed for the HANA agents. These strategies can be combined resulting in a dynamic negotiation strategy adjustable to the current available information and left time until the deadline is reached. The evolutive nature of the architecture is empowered by a search&negotiation technique. This technique corresponds to the use of the best ranked plans found by the plan search to generate the negotiation options being supplied to the negotiation strategy. And also, to the use of the information obtained from the negotiation to prune the search space and fine tune the evaluation of plans. The architecture is extensible allowing the incorporation of behavioural models that can trigger the agent intentions and affect the negotiation strategy that is guided by intentions. In fact, the world model of the agent is based on a graded BDI to simplify the incorporation of external models. Degrees are used to represent the uncertainty on the world that is basically related to the future actions to be performed by the other agents. A good usage of the search&negotiation technique reduces that uncertainty facilitating a successful performance.

Testing an architecture like HANA is now possible thanks to the testbed that we provide in this thesis. DipGame is based on The Diplomacy Game, a popular board game. We have specified the game in Z-Notation formalism, and we provide a representation of it as an RNP. HANA agents can become software players of this game, and compete against humans. The testbed consists of a complete infrastructure for the development of software agents and the execution of experiments. A language hierarchy has been defined with increasing levels of complexity. The testbed also provides tools to assist the analysis of experimental results. An experiment with negotiating and non negotiating agents is included in this thesis.

Finally, we complete the thesis applying the testbed to the game industry. We have developed and deployed a web application for playing Diplomacy online against software agents. It provides a map representing the game state, and a chat to allow the negotiation with the other players. A library for a (restricted set of) English translation to the formal agent language has been created in order to facilitate the communication among human and software agents. This application is hosted by the DipGame website where all the documentation and resources on the testbed are published. Moreover, the DipGame testbed has been integrated with Droidippy, that is a mobile solution for playing Diplomacy.

In the following, we compare each of our contributions with the related work included in Chapter 2.

First - Designing an agent architecture The principal difference between HANA and other architectures is its extensibility facilitating the incorporation of other research models, and its applicability to the industry. This is not only a theoretical work providing a new search&negotiation technique. It is also a practical work, [Lopes et al., 2008], that takes into account the complexity of searching for proposals, and the uncertainty in the environment. It includes pre-negotiation that is necessary as refers to the study of negotiation opponents, the generation of possible negotiating options, and the planning of a strategy for the negotiation. Contrarily to other works allowing multiple bilateral negotiation, HANA is designed as a single negotiating agent (in [Rahwan et al., 2002; Kraus, 1995a] a negotiator is itself a MAS), and it is able to firmly commit to the performance of actions. It does not use protocols including decommitment to avoid possible agreement overlap like in [Nguyen and Jennings, 2005].

Comparing HANA with LOGIC, [Sierra and Debenham, 2007]: HANA is simpler than LOGIC, more detailed, and computationally realistic. LOGIC is general, it describes a theoretical approach to build very intelligent software agents with negotiation capabilities suitable for taking part of a wide kind of negotiations, also those involving humans. Consequently, it does not give intuitive details that could be easily followed to build a software agent with those capabilities. HANA has been inspired by LOGIC, and it shares the same interest on pre-negotiation. Due to the fact that the architecture is specially designed for the RNP, a description of how to formulate proposals can be included in HANA. LOGIC, as most negotiation models, assumes that all possible negotiation partners have similar capabilities, and thus, selecting a partner does not depend on the concrete proposal to make. Here, instead, proposals determine which agents we should interact with as the capabilities of the negotiation partners can be different.

Second - Developing a testbed The DipGame testbed allows the adjustment of the negotiation language and protocol allowing to perform bilateral, multiple bilateral, and multilateral negotiation. There is place for information sharing, explanation, argumentation, and the enactment of emotions. DipGame is richer than the testbeds described in Section 2.3. That is so, mainly, because of the richness of the language being proposed, but also due to the possibility of observing public (the moves) and private (the offers) behaviour of agents, and due to the long duration of a game. None of the related testbeds is rich enough to allow for the testing of those complex negotiation models that can be tested in DipGame. For instance, those incorporating pre-negotiation. We encourage research on negotiating agents to be capable to interoperate with people, and the other testbeds goal, mostly, follows a constructivist approach. [Collins et al., 1998] and [Wellman and Wurman, 1999] focus on multilateral negotiations, and the ART testbed [Fullam et al., 2005] on trust and reputation. The Colored Trails Game, [Gal et al., 2005], uses an abstract game that requires the instruction of human players before taking part of the experimentation. It also requires of an initialisation that has a huge impact to the outcome of the experimentation. The use of Diplomacy in DipGame establishes an initialisation, and benefits from the popularity of the game in order to congregate human players to take part in experiments. GENIUS, [Lin et al., 2011], that is the testbed used by the Automated Negotiating Agent Competition (ANAC), assumes that the negotiation domain is known by the agents and static. The negotiation is bilateral using an alternating protocol. In the competition, the agent s preferences are known, represented as utility functions, and fixed during the whole negotiation session. And the issues are independent. As a consequence, the ANAC competition is quite limited to a particular negotiation domain, and negotiation strategies. DipGame provides more flexibility for those strategies to contemplate a more realistic problem.

Third - Problem formalisation A Resource Negotiation Problem is a resource allocation problem that allows negotiation about actions to be performed individually, but simultaneously, by a group of agents. We think that it is more realistic than other problems proposed in the automated negotiation research field, like those used in the ANAC competition. Negotiating about actions is quite different to negotiating about numerical values to assign to particular issues. The utility provided by the actions is rarely independent. Moreover, their benefit for an agent depends a lot on the agent mental state and the current situation. The agents utilities cannot be fixed during the negotiation processes.

Fourth - Specification of The Diplomacy Game In this thesis, we provide a formal specification of The Diplomacy Game that was lacking before. Previous work describing the adjudicator algorithm that resolves the game was not formal

[Black et al., 1998]. The existing documentation on the game relates to the rules, and the study of tactics and openings [Sharp, 1978b,a; Kruijswijk, 2009]. This specification and the complexity analysis of the game are necessary for the use of The Diplomacy Game as a domain for experimentation.

Fifth - Application to online game industry To our knowledge there is no previous negotiation testbed that was applied to the industry. The DipGame application and the collaboration with Droidippy, corroborate our aim of doing practical research with real potential applications. This work is a success thanks to the multiple users logged in DipGame, and their activity playing against our software agents.

8.2 Future work

In the future, we would like to study the use of a personality model, like the one presented in [Pena et al., 2011], to select how agents should emotionally react to observations by generating appropriate intentions. Analysing the personality of other agents is specially challenging as our agent counterparts can be humans. Another interesting model to incorporate is the relationship model described in [Sierra and Debenham, 2007]. In it, intimacy levels are computed for each other agent, and some strategies to reach a certain desired intimacy level are provided. The next behaviour model we plan to extend the architecture with is a trust model. The concept of trust is really important to perform negotiations. In environments where there is no trust among agents, negotiation cannot take place as the negotiators will not be able to belief on the actual execution of the agreement. We plan to incorporate a version of Sierra and Debenhams trust model [Sierra and Debenham, 2006]: (i) to provide beliefs on the agent s trust on other agents; (ii) to update the belief set from reached commitments; (iii) to provide the intentions to negotiate or not with a given agent; and (iv) to exclude from the plan ranking those joint plans with actions assigned to agents whom the HANA agent distrusts.

Apart from that, the future work by contribution would be the following:

HANA Something that is lacking in the architecture and that we would like to work further are the internal details of the world model that is represented as a g-BDI with dynamic desires and emotions. The work done in [Casali et al., 2011] and [Joseph, 2010] is crucial to this end. We would like to provide an implementation of the architecture as a software framework where the world model content, also the transition functions between beliefs, desires and intentions, were input data. Such software would facilitate a lot the development of HANA agents, specially those to run on DipGame thanks to the framework for developing software agents that it already provides.

DipGame The testbed is in production, and available at the DipGame website. There is a tutorial, and other documentation describing how to use it, and it is being used by other students. The use of the testbed, as it is today, by other research labs is proved to be possible. Despite of that, it seems that it is not as simple as, for instance, GENIUS. By problem definition, the design and implementation of an agent for DipGame must be more difficult than an agent for GENIUS is (more complex and richer problem implies more difficult agent design). A DipGame agent will always be more difficult to create than a GENIUS agent. However, some improvements can be done in DipGame to simplify this task. The existence of the before mentioned implementation of HANA would simplify the creation of agents. Another option is to provide a set of implemented Diplomacy strategies. Those are currently described in the Diplomacy communities, but there is no publicly available implementation of them.

Application Another way to motivate players to engage in new games is the incorporation of a global ranking of players in the web. In the future, we would like to do this taking advantage of the work done in [Osman et al., 2010]. Human players, as well as software players, would be ranked. A ranking of those players would challenge their improvement. Other testbeds organise competitions that take place in workshops and conferences, [Wellman and Wurman, 1999]. We rather plan a continuous competition enforced by the ranking.

8.3 Related publications

The work described in this thesis has been published in several journals and in the main MAS conference proceedings. It has also been presented in several other conferences with proceedings and in several workshops. Some presentations included video and direct demonstrations of the testbed software. Next, we list the publications indicating the url of the video demonstrations when available:

Angela Fabregues and Carles Sierra. HANA: a Human-Aware Negotiation Architecture. *Decision Support Systems*, In press.

Angela Fabregues, Santiago Biec and Carles Sierra. Running Experiments on DipGame Testbed (Demonstration). In *Proc. of the 11th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2012).*, pages , Valencia, 04/06/2012 2012. `http://www.dipgame.org/media/AAMAS2012demo`

Angela Fabregues and Carles Sierra. DipGame: a challenging negotiation testbed. *Engineering Applications of Artificial Intelligence*, Vol. 24(7), pages 1137 1146, 2011.

Angela Fabregues, David López-Paz and Carles Sierra. DipTools: Experimental Data Visualization Tool for the DipGame Testbed (Demonstration). In *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, pages 1315 1316. `http://www.dipgame.org/media/AAMAS2011demo`

Angela Fabregues and Carles Sierra. An Agent Architecture for Simultaneous Bilateral Negotiations In *Proc. of the 13è Congrés Int. de l'Associació Catalana d'Intel.ligència Artificial (CCIA 2010)*, pages 29 38.

Angela Fabregues, David Navarro, Alejandro Serrano and Carles Sierra. DipGame: a Testbed for Multiagent Systems (Demonstration) In *Proc. of 9th Int. Conf.*

on Autonomous Agents and Multiagent Systems (AAMAS 2010), pages 1619 1620. `http://www.dipgame.org/media/AAMAS2010demo`

Appendix A

Quick start guide

Learn the basics to develop your own software player (bot) with dipGame and run Diplomacy games against it.

A.1 Introduction

Diplomacy is a game played by seven players. In dipGame we want to create software players of this game and make them compete along them and against human players. DAIDE was a pioneer doing it. We continue their work and we made it more suitable for scientific researchers of areas like Multiagent Systems.

In the following we assume that you know the rules of the game and refer to spring movements as SPR, spring retreates as SUM, fall movements as FAL, fall retreates as AUT and adjustments as WIN. The available orders in each season are (for movements:) to hold, move, support hold, support move, (for retreates:) retreat, disband, (for adjustments:) build, remove and waive. Convoys are not allowed in dipGame.

A.2 Play your first game

Every user can play Diplomacy on-line using our website (click on New Game).

If you intend to create your own bot, you should learn how to play off-line, on your own computer. To do so: download the Game Manager , install it and run a game involving 6 bots and a human player.

A.3 Create your first bot

The complexity of creating a bot varies a lot depending on your requirements. The simplest bot ever just holds its position, disbands and removes units or waives depending on the season of the game. We call this bot HoldBot.

To implement HoldBot we can extend the `es.csic.iiia.fabregues.dip.Player` class from the dip framework and implement its method `play()`. Other method headers must be also in our `HoldBot` class (like `receivedOrder(Orderarg0)`) but it is not necessary to include any sentence into them. The method `play()` for HoldBot should look like the following:

```
@Override
 public  List<Order> play() {
  List<Order> orders = new  Vector<Order>();
  switch  (game.getPhase()) {
  case  SPR:
  case  FAL:
   //Holding all controlled units
   for (Region unit: me.getControlledRegions()){
    HLDOrder hldOrder = new  HLDOrder(me, unit);
    orders.add(hldOrder);
   }
   break ;
  case  SUM:
   case  AUT:
      //Disbanding all dislodged units
      for (Region dislodgedUnit: game.getDislodgedRegions(me)){
        DSBOrder dsbOrder = new  DSBOrder(dislodgedUnit, me);
        orders.add(dsbOrder);
      }
      break ;
    default :
      //That's WIN
      int  nBuilds = me.getOwnedSCs().size()-me.getControlledRegions().
          size();
      if (nBuilds > 0){
        //Waiving nBuilds times
        for (int  i=0; i<nBuilds; i++){
          WVEOrder wveOrder = new  WVEOrder(me);
          orders.add(wveOrder);
        }
      }else if (nBuilds < 0){
        //Removing nBuilds units
        int  nRemovals = -nBuilds;
        for (int  i=0; i<nRemovals; i++){
          Region remUnit = me.getControlledRegions().get(i);
          REMOrder remOrder = new  REMOrder(me, remUnit);
          orders.add(remOrder);
        }
      }
      break ;
    }
    return  orders;
  }
```

The HoldBot as described before is not a program because it has no main method. To complete it and be able to execute your bot you need to define a method like the following one:

```
public static  void  main(String[] args){
    try  {
      IComm comm = new  DaideComm(InetAddress.getByName("localhost"),
          16713, "HoldBot");
      HoldBot holdBot = new  HoldBot();
      holdBot.start(comm);
    } catch  (UnknownHostException e) {
      System.err.println("Unknown host name.");
    } catch  (CommException e) {
      System.err.println("Cannot connect to the server.");
    }
  }
```

dip requires three libraries. Thus, your bot requires a total of four libraries to be imported. Copy those libraries into a folder together with the HoldBot.java file. Run a terminal and locate yourself into the same folder. To compile your bot you just need to type the following sentence:

```
javac -cp .:dip-1.6.jar:tcpIpComm-0.1.2.jar:jac-0.8.1.jar:utilities
```

```
    -1.0.3.jar HoldBot.java
```

If the compilation works fine no output must apear. Now, your HoldBot is ready to play games.

A.4 Play against your bot

To play a game against your new HoldBot you can run a game with gameManager setting as empty one of the players. Then, a game will be launched with several players connected to it and you should run your HoldBot connecting to the same server location. If you want to run HoldBot in the same computer that you are running the gameManager, then the game server ip is localhost and the server port is 16713 , that are the same values that we specified in the main method. Now you can execute your bot as follows:

```
java -cp .:dip-1.6.jar:tcpIpComm-0.1.2.jar:jac-0.8.1.jar:utilities
    -1.0.3.jar HoldBot
```

You can also add your bot to gameManager as described at the gameManager page. To do so it is recommended to generate an executable jar file containing your code and the required libraries. There are several ways to do so, the easiest using Apache Ant.

A.5 Add negotiation capabilities to your bot

Adding negotiation capabilities means adding a new level of complexity to the development of your bot. We recommend to exercise first developing bots without negotiation capabilities.

The easiest negotiator bot is the one that talks about peace and alliance without understanding at all what do they mean. This bot proposes randomly peaces and alliances and accepts or rejects them also randomly. This bot is useless as a negotiator but it is probably the best way to exemplify the development of a negotiating bot.

To negotiate in dipGame we use the nego negotiation framework. nego provides the class `org.dipgame.negoClient.simple.DipNegoClientImpl` for sending messages to other Diplomacy players and the class `org.dipgame.negoClient.simple.DipNegoClientHandler` to indicate what to do whenever the bot receives a message.

In the following example we create a new class `RandomNegotiator` implementing the `org.dipgame.negoClient.Negotiator` interface that creates and runs the negotiation. Then, we create a new bot extending HoldBot and adding the capability of negotiating with the previous random negotiator:

```
import java.net.InetAddress;
import java.util.List;

import org.dipgame.examples.negotiator.Negotiator;
import org.dipgame.examples.negotiator.RandomNegotiator;

import es.csic.iiia.fabregues.dip.orders.Order;
```

```
public class RandomNegotiatorHoldBot extends HoldBot{
  private int negotiationPort;
  private InetAddress negotiationServer;
  private Negotiator negotiator;

  public RandomNegotiatorHoldBot(InetAddress negotiationIp, int
      negotiationPort){
    super();
    this.negotiationServer = negotiationIp;
    this.negotiationPort = negotiationPort;
  }

  @Override
  public void init() {}

  @Override
  public void start() {
    super.start();
    negotiator = new RandomNegotiator(negotiationServer, negotiationPort
        , this);
    negotiator.init();
  }

  @Override
  /**
   * Negotiates before sending orders
   */
  public List<Order> play() {
    negotiator.negotiate();
    return super.play();
  }
}
```

RandomNegotiator negotiates only about peace and alliance but the negotiation can include moves and other sort or offers.

A.6 Template for a negotiating bot

As a start point for developing a negotiating bot we recommed you to use the following template of an Eclipse project.

Mybot-1.0.zip Mybot-1.0.tar

This template is useful also if you don t use Eclipse as its readme.txt file explains how to compile and execute it in both cases. It also explains how to generate an executable jar file from the code.

Appendix B

Notation summary

Chapters 3 and 5 contains a formalisation of the RNP and the HANA architecture that requires the definition of many concepts. In Table B.1 we summarise the notation symbols used by concept in order to assist the reader in case of confusion.

concept	set/function	element/value
state transition system	Ω	
negotiation illocutionary particles	Θ	θ
history	H	
state, first state	W, W_H	ω, ω_0
agents	$\mathcal{A}$	α, β
operator	Op	op_0, op_1
action	A	a, b
plan	P	p, q
feasible plans	P^ω	p, q
complete plans	$\bar{P}^\omega$, $\bar{P}^\omega_\alpha$	p, q
option	$\mathcal{O}^\omega$, $\mathcal{O}^\omega_\alpha$	δ
compatibility	compt(a, b)	{true, false}
feasible	feasible(a, ω), feasible(p, ω)	{true, false}
state transition function	$\mathbf{T} : W \times P \to W$	ω
expected state transition function	$\mathbb{T}^t_\alpha(\omega' \mid \omega, p)$	$[0, 1]$
commitment	C^Ψ, $C_{i\alpha} = C^{\Psi_{i\alpha}}$	c
negotiation dialogue	Ψ, Ψ_H, Ψ_α, $\Psi_{\alpha\beta}$, Ψ_i, $\Psi_{i\alpha}$, $\Psi_{i\alpha\beta}$	
utterance	M, M^ω	μ
state utility	$\mathcal{U}^t_\alpha : W \to [0, 1]$	
plan utility	$\mathcal{U}^t_\alpha : W \times P \to [0, 1]$	
plans and messages	$\Phi = P \cup M$	φ
beliefs	$\mathcal{B}$, $\mathcal{B}_\alpha$, $\mathcal{B}^t_\alpha$, $\mathcal{B}^t_\alpha : \Phi \to [0, 1]$	$\langle \varphi, \vartheta_\varphi \rangle$
agent belief review function	$\sigma_\alpha : \mathcal{B}_\alpha \times \mathcal{B}_\alpha \to \mathcal{B}_\alpha$	
confidence	$\mathcal{C}^t_\alpha : P \to [0, 1]$	
trust	$\mathcal{T}^t_\alpha : A \to [0, 1]$, $\mathcal{T}^t : \mathcal{A} \to [0, 1]$	

Table B.1: Symbols used in this document.

Bibliography

jdip, a diplomacy mapper and adjudicator, 2004. `http://jdip.sourceforge.net/`.

The diplomatic pounch. `http://www.diplom.org`.

List of diplomacy turnaments. `http://www.diplom.org/Face/cons`.

The diplomatic pouch zine. `http://devel.diplom.org/Zine/`.

J. S. Adams. Inequity in social exchange. In L. Berkowitz, editor, *Advances in experimental social psychology*, volume 2. New York: Academic Press, 1965.

C. E. Alchourròn, P. Gardenfors, and D. Makinson. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50:510 530, 1985.

Teresa Alsinet, Carlos Chesnevar, Lluís Godo, and G. Simari. A logic programming framework for possibilistic argumentation: Formalization and logical properties. *Fuzzy Sets and Systems*, 159(10):1208 1228, 2008.

Ramon Alvarez-Valdes, Enric Crespo, and Jose M Tamarit. Design and implementation of a course scheduling system using tabu search. *European Journal of Operational Research*, 137(3):512 523, 2002. ISSN 0377-2217. doi: 10.1016/S0377-2217(01)00091-1. URL `http://www.sciencedirect.com/science/article/pii/S0377221701000911`.

Leila Amgoud and Henri Prade. Using arguments for making and explaining decisions. *Artificial Intelligence*, 173(3-4):413 436, 2009.

Carles Sierra Angela Fabregues, Santiago Biec. Running experiments on dipgame testbed (demonstration). In *Proc. of the 11th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, 2012. URL `http://www.iiia.csic.es/files/pdfs/aamas2012demo-cameraReady.pdf`.

Victor Bardadym. Computer-aided school and university timetabling: The new wave. In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*, pages 22 45. Springer Berlin / Heidelberg, 1996. ISBN 978-3-540-61794-5. URL `http://dx.doi.org/10.1007/3-540-61794-9_50`.

M. H. Bazerman, G. F. Loewenstein, and S. B. White. Reversal of preference in allocation decisions: judging an alternative versus choosing among alternatives. *Administration Science Quarterly*, (37):220 240, 1992.

Martin Bichler, Gregory Kersten, and Stefan Strecker. Towards a structured design of electronic negotiations. *Group Decision and Negotiation*, 12:311 335, 2003. ISSN 0926-2644.

Kristie Black, Thaddeus Black, Brandon Clarke, Bob Dengler, Bogdan Florescu, Cait Glasson, Manus Hand, David Lawler, Gary Pennington, Ray Setzer, Simon Szykman, Tarzan, and Sandy Wible. The diplomacy player s technical guide. 1998.

Rafael H. Bordini, Michael Wooldridge, and Jomi Fred Hubner. *Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology)*. John Wiley & Sons, 2007. ISBN 0470029005.

Tibor Bosse and Catholijn M. Jonker. Human vs. computer behaviour in multi-issue negotiation. In *Proceedings of the Rational, Robust, and Secure Negotiation Mechanisms in Multi-Agent Systems (RRS'05) on Multi-Agent Systems*, pages 11 24, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2480-X. doi: 10.1109/RRS.2005.8. URL `http://dl.acm.org/citation.cfm?id=1114694.1115279`.

Jonathan Bowen. *Formal Specification and Documentation Using Z: A Case Study Approach*. International Thomson Computer Press, February 1996. ISBN 1850322309. URL `http://www.zuser.org/zbook/`.

Michael E. Bratman. *Intention, Plans, and Practical Reason*. Cambridge University Press, March 1999. ISBN 1575861925.

Jay Burmeister and Janet Wiles. The challenge of go as a domain for ai research: A comparison between go and chess. In *Third Australian and New Zealand Conference on Intelligent Information Systems (ANZIIS-95)*, pages 181 186, 1995.

Allan Calhamer. The invention of diplomacy. *Games & Puzzles*, 21, January 1974. URL `\url{http://www.diplom.org/~diparch/resources/calhamer/invention.htm}`.

Ana Casali, Lluís Godo, and Carles Sierra. A graded bdi agent model to represent and reason about preferences. *Artificial Intelligence*, 175:1468 1478, 2011.

John Collins, Maksim Tsvetovat, Bamshad Mobasher, and Maria Gini. Magnet: A multi-agent contracting system for plan execution. In *Proc. of Artificial Intelligence and Manufacturing Workshop*, pages 63 68, 1998.

Rosaria Conte and Jaime Sichman. Depnet: How to benefit from social dependence. *Journal of Mathematical Sociology*, 20(2-3):161 177, 1995.

Giorgio Coricelli and Rosemarie Nagel. Neural correlates of depth of strategic reasoning in medial prefrontal cortex. *Proc. of the National Academy of Sciences (PNAS): Economic Sciences*, 106(23):9163 9168, 2009.

Mehdi Dastani. 2apl: a practical agent programming language. *Autonomous Agents and Multi-Agent Systems*, 16(3):214 248, 2008.

Dave de Jonge. Optimizing a diplomacy bot using genetic algorithms, 2010.

John Debenham and Carles Sierra. An agent supports constructivist and ecological rationality. In *Proc. 2009 IEEE/WIC/ACM Int. Conf. on Intelligent Agent Technology*, pages 255 258, Milano, 2009.

Sebastian Deterding, Dan Dixon, Rilla Khaled, and Lennart Nacke. From game design elements to gamefulness: defining gamification . In *Proc. of the 15th Int. Academic MindTrek Conf.: Envisioning Future Media Environments (MindTrek '11)*, pages 9 15, New York, NY, USA, 2011. ISBN 978-1-4503-0816-8.

M. d Inverno and M. Luck. Engineering agentspeak(l): A formal computational model. *Logic and Computation*, 8(3):233 260, 1998.

Mark d Inverno, Michael Luck, Michael Georgeff, David Kinny, and Michael Wooldridge. The dmars architecture: A specification of the distributed multi-agent reasoning system. *Autonomous Agents and Multi-Agent Systems*, 9: 5 53, 2004. ISSN 1387-2532. URL `http://dx.doi.org/10.1023/B:AGNT.0000019688.11109.19`.

Mark d Inverno, Michael Luck, Pablo Noriega, Juan A. Rodríguez-Aguilar, and Carles Sierra. Communicating open systems. *Artificial Intelligence*, 03/2012 2012. URL `http://www.sciencedirect.com/science/article/pii/S0004370212000252?v=s5`.

Carmel Domshlak, Eyke Hullermeier, Souhila Kaci, and Henri Prade. Preferences in AI: An overview. *Artificial Intelligence*, 175(7-8):1037 1052, 2011. URL `http://www.sciencedirect.com/science/article/B6TYF-52M9N6S-2/2/1b340448ead4f69710e0ccfc17ffd722`.

C. Eisenegger, M. Naef, R. Snozzi, M. Hienrichs, and E. Fher. Sequence of testosterone vs. placebo on ultimatum game behavior in women. *Nature*, (463):356 359, 2010.

Eithan Ephrati. Strategy-planning an optimal strategy for a diplomacy player, 1987.

Angela Fabregues and Carles Sierra. An agent architecture for simultaneous bilateral negotiations. In *Proc. of the 8th European Workshop on Multi-Agent Systems*, Paris, 16/12/2010 2010.

Angela Fabregues and Carles Sierra. Diplomacy game: the test bed. *PerAda Magazine*, 2009.

Angela Fabregues, Jordi Madrenas, Carles Sierra, and John Debenham. Supplier performance in a digital ecosystem. In *Proc. of the IEEE Int. Conf. on Digital Ecosystems and Technologies (IEEE-DEST 2009)*, pages 466 471, Istanbul, 01/06/2009 2009.

Angela Fabregues, David Navarro, Alejandro Serrano, and Carles Sierra. Dipgame: a testbed for multiagent systems (demonstration). In *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 1619 1620, 2010.

Angela Fabregues, David López-Paz, and Carles Sierra. Diptools: Experimental data visualization tool for the dipgame testbed (demonstration). In *10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, pages 1315 1316, Taipei, Taiwan, 02/05/2011 2011. URL `http://www.dipgame.org`.

P. Faratin, C. Sierra, and N. R. Jennings. Negotiation Decision Functions for Autonomous Agents. *Int. Journal of Robotics and Autonomous Systems*, 24(3-4): 159 182, 1998a. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.9.1585`.

Peyman Faratin, Carles Sierra, and Nicholas R. Jennings. Negotiation decision functions for autonomous agents. *Robotics and Autonomous Systems*, 24(3-4): 159 182, 1998b.

D. Fessler and K. J. Haley. *Genetic and cultural evolution of cooperation*, chapter The strategy of affect: emotions in human cooperation, pages 7 36. 2003.

Karen K. Fullam, Tomas B. Klos, Guillaume Muller, Jordi Sabater, Andreas Schlosser, Zvi Topol, K. Suzanne Barber, Jeffrey S. Rosenschein, Laurent Vercouter, and Marco Voss. A specification of the agent reputation and trust (art) testbed: experimentation and competition for trust in agent societies. In *Proc. of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 512 518, Utrecht, The Netherlands, 2005.

Ya akov Gal, Barbara J. Grosz, Sarit Kraus, Avi Pfeffer, and Stuart Shieber. Colored trails: A formalism for investigating decision-making in strategic environments. In *IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games*, pages 25 30, 2005.

Ya akov Gal, Barbara Grosz, Sarit Kraus, Avi Pfeffer, and Stuart Shieber. Agent decision-making in open mixed networks. *Artificial Intelligence*, 174:1460 1480, December 2010.

Fredrik Håård. Multi-agent diplomacy - tactical planning using cooperative distributed problem solving. Master s thesis, Blekinge Institute of Technology, 2004.

David Hales. Cpm-03-109: Neg-o-net a negotiation simulation test-bed. Technical report, Center for Policy Modelling, 2002.

Sven Ove Hansson. *A Textbook of Belief Dynamics: Solutions to Exercises.* Kluwer Academic Publishers, Norwell, MA, USA, 2001. ISBN 0792353269.

K. Hindriks, M. d Inverno, and M. Luck. Architecture for agent programming languages. In *Proc. of the 14th European Conf. on Artificial Intelligence (ECAI 2000)*, pages 363 367, 2000.

Koen Hindriks, Catholijn M. Jonker, Sarit Kraus, Raz Lin, and Dmytro Tykhonov. Genius: negotiation environment for heterogeneous agents. In *Proc. of the Eighth International Conference on Autonomous Agents and Multiagent Systems*, pages 1397 1398, Richland, SC, 2009. IFAMAS.

Nicholas R. Jennings, Peyman Faratin, A. R. Lomuscio, Simon Parsons, Michael Wooldridge, and Carles Sierra. Automated negotiation : prospects, methods and challenges. *Group Decision and Negotiation*, 10(2):199 215, 2001.

Lauren Keller Johnson and Richard Luecke, editors. *The essentials of negotiation.* Business literacy for HR professionals series. Harvard Business School Press [u.a.], Boston, Mass. [u.a.], 2005. ISBN 1591395747. URL `http://gso.gbv.de/DB=2.1/CMD?ACT=SRCHA&SRT=YOP&IKT=1016&TRM=ppn+392891115&sourceid=fbw_bibsonomy`.

Sindhu Joseph. Coherence-based computational agency, April 2010.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101: 99 134, May 1998.

Markus Kemmerling, Niels Ackermann, Nicola Beume, Mike Preuss, Sebastian Uellenbeck, and Wolfgang Walz. Is human-like and well playing contradictory for diplomacy bots? In *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG 2009)*, pages 209 216, 2009.

Markus Kemmerling, Niels Ackermann, and Mike Preuss. Nested look-ahead evolutionary algorithm based planning for a believable diplomacy bot. In *Applications of Evolutionary Computation - EvoApplications 2011, Part I, Lecture Notes in Computer Science (LNCS)*, pages 83 92, 2011.

Markus Kemmerling, Niels Ackermann, and Mike Preuss. Individualizing planning diplomacy bots, 2012.

S. Kraus. *Strategic Negotiation in Multiagent Environments.* MIT Press, 2001.

Sarit Kraus. Designing and building a negotiating automated agent. *Computational Intelligence*, 11:132 171, 1995a.

Sarit Kraus. Designing and building a negotiating automated agent. *Computational Intelligence*, 11:132 171, 1995b.

Sarit Kraus, D. Lehmann, and E. Ephrati. An automated diplomacy player. In D. Levy and D. Beal, editors, *Heuristic Programming in Artificial Intelligence: The 1st Computer Olympia*, pages 134 153. Ellis Horwood Limited, 1989.

Sarit Krauss. Planning and communication in a multi-agent environment, 1988.

Lucas Kruijswijk. The math of adjudication. *The Zine*, Spring 2009 Movement, 2009. `http://www.diplom.org/Zine/S2009M/Kruijswijk/DipMath_Chp1.htm`.

Cuihong Li, Joseph Giampapa, and Katia Sycara. Bilateral negotiation decisions with uncertain dynamic outside options. In *Proceedings of the First IEEE International Workshop on Electronic Contracting*, WEC 04, pages 54 61, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2184-3.

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*, 2011.

Daniel E. Loeb and Michael R. Hall. Thoughts on programming a diplomat. Technical Report 90-108, UNIVERSITE DE BORDEAUX 1. Talence, 1990. URL `http://opac.inria.fr/record=b1047926`.

Fernando Lopes, Nuno Mamede, A. Q. Novais, and Helder Coelho. A negotiation model for autonomous computational agents: Formal description and empirical evaluation. *J. Intell. Fuzzy Syst.*, 12(3,4):195 212, December 2002. ISSN 1064-1246.

Fernando Lopes, Michael Wooldridge, and Augusto Q. Novais. Negotiation among autonomous computational agents: principles, analysis and challenges. *Artif. Intell. Rev.*, 29(1):1 44, 2008. URL `http://dblp.uni-trier.de/db/journals/air/air29.html#LopesWN08`.

Michael Luck and Peter McBurney. Computing as interaction: Agent and agreement technologies. pages 1 6, 2008.

Marvin Minsky. The emotion machine: from pain to suffering. In *Proc. of the 3rd Conf. on Creativity & cognition (C&C '99)*, pages 7 13. ACM, 1999.

Steve Munroe. Motivation and autonomy for pre-negotiation, September 2005.

Dirk Neumann, Morad Benyoucef, Sarita Bassil, and Julie Vachon. Applying the montreal taxonomy to state of the art e-negotiation systems. *Group Decision and Negotiation*, 12:287 310, 2003. ISSN 0926-2644.

T.D. Nguyen and N. R. Jennings. Managing commitments in multiple concurrent negotiations. *International Journal Electronic Commerce Research and Applications*, 4(4):362 376, 2005. URL `http://eprints.soton.ac.uk/260817/`.

Thuc Duong Nguyen and Nicholas R. Jennings. Coordinating multiple concurrent negotiations. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS 04, pages 1064 1071, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 1-58113-864-4.

David Norman. Diplomacy ai development environment message syntax, 2006. `http://www.ellought.demon.co.uk/dipai/dpp_syntax.rtf`.

Nardine Osman, Carles Sierra, and Angela Fabregues. A propagation and aggregation algorithm for inferring opinions in structural graphs. In *Thirteenth International Workshop on: trust in agent societies, at AAMAS 2010*, Toronto, Canada, 10/05/2010 2010.

Pere Pardo, Pilar Dellunde, and Lluís Godo. Argumentation-based negotiation in t-delp-pop. volume 232, pages 177 186. IOS Press, IOS Press, 2011. ISBN 978-1-60750-841-0.

Simon Parsons and Paolo Giorgini. An approach to using degrees of belief in BDI agents. In *Proc. of the Int. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, 1999.

Luis Pena, Jose-María Pena, and Sascha Ossowski. Representing emotion and mood states for virtual agents. In *Proc. of the 9th German Conf. on Multiagent system technologies*, pages 181 188, 2011.

Pablo Pilotti, Ana Casali, and Carlos Chesnevar. An approach to automated agent negotiation using belief revision. In *Proc. of the12th Argentine Symposium on Artificial Intelligence (40th JAIIO)*, pages 202 212, Córdoba, Argentina, 2011.

Jr. Pinho, Orlando, Geber Ramalho, Gustavo Paula, and Patrícia Tedesco. Sequential bilateral negotiation. In AnaL.C. Bazzan and Sofiane Labidi, editors, *Advances in Artificial Intelligence - SBIA 2004*, volume 3171 of *Lecture Notes in Computer Science*, pages 526 535. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-23237-7.

Sylwia Polberg, Marcin Paprzycki, and Maria Ganzha. Developing intelligent bots for the diplomacy game. In *Federated Conference on Computer Science and Information Systems (FedCSIS'11)*, pages 589 596, 2011.

Iyad Rahwan, Ryszard Kowalczyk, and Ha Hai Pham. Intelligent agents for automated one-to-many e-commerce negotiation. In *Proceedings of the twenty-fifth Australasian conference on Computer science - Volume 4*, ACSC 02, pages 197 204, Darlinghurst, Australia, Australia, 2002. Australian Computer Society, Inc. ISBN 0-909925-82-8. URL `http://dl.acm.org/citation.cfm?id=563801.563824`.

H. Raiffa. *Negotiation Analysis: The Science and Art of Collaborative Decision Making*. Harvard U.P., 2002.

Warren Raisch. *The eMarketplace: Strategies for Success in B2B eCommerce*. McGraw-Hill, Inc., New York, NY, USA, 2002. ISBN 0071380124.

Anand S. Rao. Agentspeak(l): Bdi agents speak out in a logical computable language. In *Proc. of the 7th European workshop on Modelling autonomous agents in a multi-agent world (MAAMAW '96)*, pages 42 55, 1996.

Anand S. Rao and Michael P. Georgeff. Decision procedures for BDI logics. *Journal of Logic and Computation*, 8(3):293 342, 1998.

Anand S. Rao and Michael P. Georgeff. Modeling Rational Agents within a BDI-Architecture. In *Proc. of the 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473 484, 1991.

William Revelle and Klaus R. Scherer. Personality and emotion. In *Oxford Companion to the Affective Sciences*. Oxford University Press, 2010.

Joao Ribeiro, Pedro Mariano, and Luís Seabra Lopes. Darkblade: A program that plays diplomacy. In Luís Lopes, Nuno Lau, Pedro Mariano, and Luís Rocha, editors, *Progress in Artificial Intelligence*, volume 5816 of *Lecture Notes in Computer Science*, pages 485 496. Springer Berlin / Heidelberg, 2009. ISBN 978-3-642-04685-8.

Alan Ritchie. Diplomacy a.i. Master s thesis, The University of Glasgow, 2003.

Andrew Rose. The diplomacy centralisation project client-server protocol, 2003. `http://www.daide.org.uk/external/comproto.html`.

JS Rosenschein and G Zlotkin. *Rules of Encounter.* MIT Press, 1998.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Pearson Education, 2003. ISBN 0137903952. URL `http://portal.acm.org/citation.cfm?id=773294`.

Alan G. Sanfey, James K. Rilling, Jessica A. Aronson, Leigh E. Nystrom, and Jonathan D. Cohen. The neural basis of economic decision-making in the ultimatum game. *Science*, 300(5626):1755 1758, June 2003.

Arvind Sathi and Mark S. Fox. Distributed artificial intelligence (vol. 2). chapter Constraint-directed negotiation of resource reallocations, pages 163 193. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1989. ISBN 0-273-08810-6. URL `http://dl.acm.org/citation.cfm?id=94079.94089`.

Harold Saunders. Prenegotiation and circum-negotiation: Arenas of the peace process, 1996.

Jaspreet Shaheed. Creating a diplomat. Master s thesis, Department of Computing, Imperial College Of Science, Technology and Medicine, 180 Queen s Gate, London, SW7 2BZ, UK, June 2004.

Melvin Shakun. Multi-bilateral multi-issue e-negotiation in e-commerce with a tit-for-tat computer agent. *Group Decision and Negotiation*, 14:383 392, 2005. ISSN 0926-2644.

Ari Shapiro, Gil Fuchs, and Robert Levinson. Learning a game strategy using pattern-weights and self-play. In *Computers and Games*, pages 42 60, 2002.

R. Sharp. *The Game of Diplomacy.* A. Barker, 1978a. ISBN 9780213166762.

Richard Sharp. *The Game of Diplomacy.* 1978b. http://www.diplom.org/ diparch/god.htm.

Carles Sierra and John Debenham. The logic negotiation model. In *Proc. of 6th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2007)*, pages 1026 1033, 2007.

Carles Sierra and John K. Debenham. Trust and honour in information-based agency. In *Proc. of the 5th Int. Conf. on Autonomous Agents and Multi-agent Systems (AAMAS 2006)*, pages 1225 1232, 2006.

Herbert A. Simon. *The sciences of the artificial (3rd ed.).* MIT Press, Cambridge, MA, USA, 1996. ISBN 0-262-69191-4.

Vernon L Smith. Constructivist and ecological rationality in economics. *The American Economic Review*, 93(3):465 508, 2003.

Harris Sondak, Margaret A. Neale, and Robin Pinkley. The negotiated allocations of benefits and burdens: The impact of outcome valence, contribution, and relationship. *Organizational Behaviour and Human Decision Processes*, (3):249 260, December 1995.

J. Michael Spivey. The fuzz type-checker for z [computer software], November 2008. (Version 3.4.1) `http://spivey.oriel.ox.ac.uk/mike/fuzz/`.

J. Michael Spivey. *The Z notation: a reference manual (second edition).* Prentice Hall International: Hemel Hempstead, England, 1992. `http://spivey.oriel.ox.ac.uk/mike/zrm`.

Katia P. Sycara. Distributed artificial intelligence (vol. 2). chapter Multiagent compromise via negotiation, pages 119 137. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1989. ISBN 0-273-08810-6. URL `http://dl.acm.org/citation.cfm?id=94079.94086`.

Rustam Vahidov. Situated decision support approach for managing multiple negotiations. In Henner Gimpel, Nicholas R. Jennings, Gregory E. Kersten, Axel Ockenfels, Christof Weinhardt, Wil Aalst, John Mylopoulos, Michael Rosemann, Michael J. Shaw, and Clemens Szyperski, editors, *Negotiation, Auctions, and Market Engineering*, volume 2 of *Lecture Notes in Business Information Processing*, pages 179 189. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-77554-6.

B. Van de Walle, S. Heitsch, and P. Faratin. Coping with one-to-many multi-criteria negotiations in electronic markets. In *Database and Expert Systems Applications, 2001. Proceedings. 12th International Workshop on*, pages 747 751, 2001.

Jason van Hal. Diplomacy ai - albert. URL `\url{https://sites.google.com/site/diplomacyai/albert}`.

Adam Webb, Jason Chin, Thomas Wilkins, John Payce, and Vincent Dedoyard. Automated negotiation in the game of diplomacy. Master s thesis, January 2008.

Michael P. Wellman and Peter R. Wurman. A trading agent competition for the research community. In *AMEC, IJCAI 1999 Workshop*, 1999.

Colin R. Williams, Valentin Robu, Enrico H. Gerding, and Nicholas R. Jennings. Negotiating concurrently with unknown opponents in complex, real-time domains. In *20th European Conference on Artificial Intelligence*, pages 834 839, August 2012. URL `http://eprints.soton.ac.uk/339064/`.

Paul D. Windsor. What s your point. *The Diplomatic Pounch Zine*, 1999. Spring 1999 Movement, `http://www.diplom.org/Zine/S1999M/Windsor/point.html`.

Michael Winikoff. Jack intelligent agents: An industrial strength platform. In *Multi-Agent Programming*, volume 15, pages 175 193. Springer, 2005.

Dajun Zeng and Katia Sycara. Bayesian learning in negotiation. *Int. J. Hum.-Comput. Stud.*, 48(1):125 141, January 1998. ISSN 1071-5819.