Automated Negotiation In The Game Of Diplomacy

Adam Webb

Jason Chin

Thomas Wilkins

John Payce

Vincent Dedoyard

January 2nd, 2008

Abstract

Diplomacy is a strategic board game with relatively simple tactics but with a rich negotiation element between players. The Diplomacy AI Development Environment provides a framework within which automated Diplomacy players can be developed. We create a simple negotiating Diplomacy player within this framework and use it to show that negotiating players outperform non-negotiating ones.

Acknowledgements

First and most of all, we would like to thank Iain Phillips for supervising our project and pointing us in the right direction. We'd also like to thank Jaspreet Shaheed for meeting with us to discuss his work in AI development in Diplomacy. Also, importantly we would like to thank the entire Diplomacy AI development community, particularly members of the DipAI forums and those who developed core parts of the DAIDE architecture i.e. David Norman, Henrik Bylund, Andrew Rose, Daniel Yule, and John Newbury.

Contents

1 Introduction **6**

2 Background **8**

2.1 Game Of Diplomacy 8

2.1.1 Nature Of The Game 8

2.1.2 Tactics and Strategies 9

2.1.3 Impact on AI Development 11

2.2 DAIDE And Associated Tools 13

2.2.1 Protocol 14

2.2.2 DAIDE Server 16

2.2.3 DAIDE Mapper 17

2.2.4 DAIDE Java AI Communication API 17

2.3 Existing AIs 19

2.3.1 DumbBot 19

2.3.2 Diplomat 20

2.3.3 BlabBot 22

2.3.4 The Israeli Diplomat 22

2.3.5 The Bordeaux Diplomat 24

2.3.6 LA Diplomat 25

3 Board Representation **26**

3.1 Addressing Provinces Correctly 26

3.2 Units and Their Owners 27

4 Tactics And Strategies **29**

4.1 Weighting algorithm 31

4.1.1 Working out the strength of a power 31

4.1.2 Working out the initial values 32
4.1.3 Blurring the values around the map 32
4.1.4 Working out the strength and competition values: . . . 35
4.1.5 Working out the final destination values: 37
4.2 Generating orders 37
4.2.1 Spring/Fall turns 37
4.2.2 Summer/Autumn turns (retreat/disband) 39
4.2.3 Winter turns (builds/disband) 39

5 Negotiation 40
5.1 Message Listener and Thread Interaction 40
5.2 The Negotiator Thread 40
5.3 Our Negotiation Strategy 41
5.3.1 Peace To All 41
5.3.2 Back-stab 43

6 Conclusion and Evaluation 44
6.1 Tests Performed and Results 44
6.2 Analysis and Conclusion 44
6.3 Evaluation 48

7 Future Work 48
7.1 Improvements to the DAIDE framework 49
7.2 Improvements to Our Code Base 49
7.2.1 Banker Paradigm 49
7.2.2 Other Extensions 50
7.3 Open-ended Improvents 51

1 Introduction

Diplomacy is a classic and well respected strategy board game currently published by Avalon Hill[1] noted for its relatively simple gameplay mechanic but rich higher level play based around negotiation between players. It is well liked by board game aficionados. For example on the board game fan site BoardGameGeek it is ranked 153rd out of 4177 games rated. [2]

The game is set in Europe at the start of the 20th century. The board is divided up into 75 named provinces (which are either inland e.g. Ukraine, sea e.g. North Atlantic or coastal e.g. Holland). 34 of the inland and coastal provinces are designated as supply centres which allow players to support a unit on the board. These units are either armies or fleets. One unit is allowed in any province at any one time with the intuitive restrictions on which provinces armies and fleets are allowed to occupy. Each player takes control of one of the seven Great Powers of the time and starts with a handful of home supply centres and the units associated with them. For example England starts with an army in Liverpool and fleets in London and Edinburgh.

Play progresses with players negotiating, secretly issuing orders to their units and then once all players have finished resolving the turn. Full discussion of what orders can be issued is quite lengthy but in essence players may either hold its position, move to a province adjacent to it's current one or support the move of another unit (this is useful as all units are of equal strength so force of numbers determines whether or not a move is successful Players advance capturing new supply centres allowing them to build new units at their home centres. If a players loses supply centres to enemy action they must disband (remove from the board) units to reflect the number of supply centres they have. Once all orders have been issued they are executed simultaneously to determine the new state of the board. Orders may be unsucessful (they are referred to as bouncing). A simple example of this is if two units attempt to move into the same province. Without support from additional units both moves are unsuccesful and the pieces remain in the same positions. A player wins when they control the majority of the supply centres (i.e. 18 or more of the 34).

On top of this relatively simple game play mechanic is a complex system of negotiation. Players are free to discuss with each other any aspect in the game either privately or publicly. The scope of these negotiations is only limited by the players themselves. Alliances may be formed, provinces may be demilitarised and complex written agreements may be drawn up. However as players issue their final orders secretly intrigue and deceit are almost inevitable. Thus players must constantly guard against a betrayal by even their closest allies.

The objective of our project is to develop an AI to play Diplomacy with some degree of automated negotiation. There is a pre-existing framework for developing such AIs called the Diplomacy AI Development Environment (DAIDE) [3]. This provides a protocol for playing the game and negotiating as well as various tools (e.g. a server which hosts games and resolves orders and a mapper which represents the game graphically and provides an interface for human players). Our AI is intended to work within this framework.

2 Background

2.1 Game Of Diplomacy

We have briefly discussed the game of Diplomacy in the introduction to this report. Full treatment of the rules of the game would be too lengthy to be usefully included here. One should refer to the official rules of the game [4] for specific discussion of the minutiae of play. This chapter is reserved for general discussion of the nature of the game as well as possible tactics and strategies for good play.

2.1.1 Nature Of The Game

It seems wise at this stage to discuss some general properties of the game of Diplomacy.

Completely Deterministic The results of a turn are completely determined by the state of the board at the start of that turn and the orders issued by players during that turn. There is no element of chance involved.

Imperfect Information The simultaneous execution of secretly determined moves means that players do not have complete knowledge about the actions of other players i.e. they have imperfect information. This means that even though the game is deterministic players cannot predict the results of their actions even one turn in advance. This is in stark contrast to games such as Chess and Go where AI development is more mature.

Zero-Sum A zero-sum game is a game in which a player's gains or losses are exactly counterbalanced by the gains or losses of other players This is true of Diplomacy. A player can gain supply centres only by denying them to other players.

Psychological Aspect The importance of the psychological aspect of diplomacy is hard to overestimate. An effective player must attempt to predict the actions of other players. A better must be able to manipulate the actions of others to suit its own ends. Furthermore players must be able to recognise when this is being done to them. This is something human players are very good at but is very difficult to program in bots. Concepts such as honesty and deceit; trust and suspicion; rewards and revenge are integral to the game and a good bot should reflect this.

2.1.2 Tactics and Strategies

Diplomacy is a very popular game with a dedicated fan base. As such there is a wealth of material discussing how to play. Notably there is a player edited wiki [5] and an archive containing a large collection of articles written over the years [6]. This material ranges from general strategy, recommended openings, end game considerations and psychological insights. This material is obviously vast with a degree of internal contradiction represnting disagreements amongst players. However there is some limited consensus among this material.

Opening Phase

The opening of the game has several notable aspects. Much is written on various opening strategies for different powers and from this one can make several general observations on what is important.

- Securing neutral supply centres is important. Negotiation should be used to achieve this with minimal conflict at this stage. This helps prevent your power from falling behind by failing to gain supply centres when others do.
- One should gain information on other players. Human players attempt

to feel each other out at the start of the game. Many guides put emphasis on communicating with every player even those who are on the other side of the board in order to acquire information about them [7]. This is of particular importance for AI play where players may have vastly different negotiating capacities.

- Observing which directions other players move in is likely to reveal which players will come into conflict during the game. For example England moving a fleet into the English Channel is widely seen as likely to bring England and France into conflict. [8]

Mid-Game

Mid-game is the most open phase of play and makes up the bulk of the game. There are several things a good player should do in this stage.

- Players should prevent opponents reaching a dominant position. Attacking the largest power may well be a good strategy in order to prevent them from achieving the momentum necessary to win.
- Attempt to bring more supply centres under our control in order to move towards victory.
- Determine which players and which are enemies. This is often determined by actions in the opening phase.
- Identify which players are our allies and avoid attacking them in preference to eliminating enemies. Avoid antagonising allies unless there is no other option.

End-Game

This is the stage where there will be one or more powers in a potential winning position. Here we should atttempt to either win if we are dominant or prevent a dominant player winning if we are not.

- If we are a small power ally with other small powers to prevent dominant powers from achieving victory.
- If we are a large power consider betraying our alliances to make the break away necessary to secure victory.

Low-Level Tactics

Above we have discussed broad considerations at the strategic level however we must use a finer level of tactics to decide what we are going to do from turn to turn. Some such considerations are:

- Units should where possible move forwards and attack enemy supply centres. If we are not near any enemy supply centres they should naturally move closer to them to make future attacks possible.
- However we should also ensure our own supply centres are not undefended.
- Units should act together in order to secure objectives. Isolated units are likely to be destroyed or pushed back.

2.1.3 Impact on AI Development

The considerations above impact on AI development in several key ways.

Large Search Tree

The number of possible moves in Diplomacy is vast This makes generating a search tree of future game states intractable. As other player's moves are secret and resolved simultaneously we cannot accurately predict the state of the game board even one turn in the future let alone deeper into the game. However our bot must not be shortsighted and consider only immediate advantage. It must be able to think about the future in general terms if not specifically.

Difficulty of Position Analysis

Unlike games such as chess where it is relatively easy to analyse the strength of a position based simply on the locations of the pieces on the board position analysis is extremely difficult in diplomacy. This is because of the psychological aspect of the game. Whilst we may have a material advantage on the board one must also consider how much we have antagonised other players. A player with many units but maby enemies is probably in a worse position than a well connected player with slightly fewer units. Heuristics to evaluate these psychological factors must be developed.

Complexity of Negotiation

Negotiation in diplomacy is extremely complex. Human players engage in a variety of different types of communication ranging from broad expressions of attitude towards each other through specific move suggestions to allegations of treachery and justifications. Thus a full negotiation language between AI's must be extremely rich. Even implementing a subset of the communication humans use is a daunting prospect.

Evaluation of the Benefits of Helping Others

A good player should recognise that helping other players now even if there is no immediate benefit could prove useful. We need a heuristic to recognise these unclear benefits based on the trustworthiness of the opposition and the value of our help.

2.2 DAIDE And Associated Tools

There are various software interfaces available that adjudicate and allow users to play games of Diplomacy.

- Online servers: There are many web-sites [9] [10] that provide users with the ability to play games on the internet via their web-site or via e-mail. Once all the users submit their orders and they have been received, the server then evaluates the moves and then sends the results to the players and the new game state.
- The Hasbro Diplomacy Game: A game developed by Microprose and released in 2000. It gave players an interface and AI to use and provided multiplayer support on local machines and the internet. AI is generally considered to be quite poor.
- DAIDE (Diplomacy AI Development Environment) [3] : This was created by the DipAI organisation. Developed to allow and encourage AI Diplomacy bots. It gives players a server, interface and tools to create an automated diplomacy player. It provides the syntax for communicating over the network for the use of negotiation. Players can also compete via local machine or the internet and there are multiple bots available [11] .

We decided to develop using DAIDE as it provides us with a server that handles communication between computers, evaluates moves, provides a graphical user interface for viewing the board and provides the syntax for players to communicate with. As it is designed specifically for AI, it will allow us to concentrate solely on creating a negotiating agent. It also provides us with a test harness against other proficient AI negotiating bots, as there are many bots available for the DAIDE server, and many of the recent bots are written for it. The DipAI community is also very active, and so they will be able to provide valuable information, advice, discussion and feedback to the project.

2.2.1 Protocol

In the DAIDE architecture, there is a central server that hosts the game, and all clients then connect to it. All communication is handled via the DAIDE server, such that if a client wants to send a message to another client, it will send the message to the server, specifying which power(s) to send it to. The server will then handle the message, check the syntax, and if everything is ok, it will send it to the specified client (or send an error message back to the client if it does not understand the message). Messages between the server and client are also sent, such as the current state of the board and the current orders.

The DAIDE syntax [12] defines communication protocol between Diplomacy players for negotiation. This was developed from the analysis between the kinds of communication that took place between players during a game. It provides a player with tokens based on those exchanges, which allow players to 'talk' between each other on a particular subject, without them having to understand natural language.

These sets of tokens are grouped in levels of press, starting from Level 0 (no press, no communication) and increasing in complexity up to Level 130 (explanations) and a special Level 8000 as free text.

Table 1 shows the press level and its definition for the DAIDE syntax hierarchy.

Using this press level hierarchy, we decided to aim for press level 20,

Level 0	No Press
Level 10	Peace and Alliances
Level 20	Order proposals
Level 30	Multipart Offers
Level 40	Sharing out the Supply Centres
Level 50	Nested Multipart Offers
Level 60	Queries and Insistences
Level 70	Requests for suggestions
Level 80	Accusations
Level 90	Future discussions
Level 100	Conditionals
Level 110	Puppets and Favours
Level 120	Forwarding Press
Level 130	Explanations
Level 8000	Free text

Table 1: Press Level & Explanation

which is generating order proposals. This includes suggesting moves and also offering peace and alliances to other players. We chose press level 20 as a bot that reaches that level will be able to instantiate relationships between other players and use that relationship in a manner that will benefit it. This will create a fairly capable bot as most other bots only play press level 0 or 10.

As there are also many different variants for the game of Diplomacy, we have chosen to play the standard map, with no time limits. This is the most basic and common variation and the one that most bots and human players play.

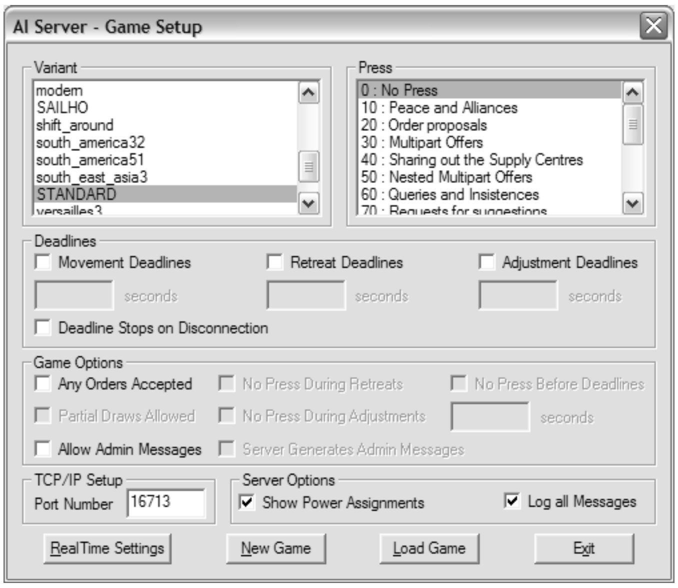
2.2.2 DAIDE Server

The DAIDE server application [13] is written by David Norman and it allows the clients that are playing or observing to connect to the Diplomacy game. It resolves the turns once everyone has submitted their orders and it also parses the messages sent throughout the game, checking for correctness.

The server allows for developers to concentrate solely on their AI, as they do not need to create their own adjudicator because the server does it for them. It also handles the syntax so that the messages that you will receive will always be of specific syntax and grammatically correct within the DAIDE language.

We will be using this server as it currently the only one available that uses the DAIDE architecture and allows us to communicate with other clients, as well allowing us to concentrate more on other aspects of our bot.

Figure 1 shows the interface for setting up a server.



AI Server - Game Setup

Variant
- modern
- SAILHO
- shift_around
- south_america32
- south_america51
- south_east_asia3
- STANDARD
- versailles3

Press
- 0 : No Press
- 10 : Peace and Alliances
- 20 : Order proposals
- 30 : Multipart Offers
- 40 : Sharing out the Supply Centres
- 50 : Nested Multipart Offers
- 60 : Queries and Insistences
- 70 : Requests for suggestions

Deadlines
- [ ] Movement Deadlines, seconds
- [ ] Retreat Deadlines, seconds
- [ ] Adjustment Deadlines, seconds
- [ ] Deadline Stops on Disconnection

Game Options
- [ ] Any Orders Accepted
- [ ] No Press During Retreats
- [ ] No Press Before Deadlines, seconds
- [ ] Partial Draws Allowed
- [ ] No Press During Adjustments
- [ ] Allow Admin Messages
- [ ] Server Generates Admin Messages

TCP/IP Setup
Port Number 16713

Server Options
- [✓] Show Power Assignments
- [✓] Log all Messages

RealTime Settings | New Game | Load Game | Exit

Figure 1: Server Interface

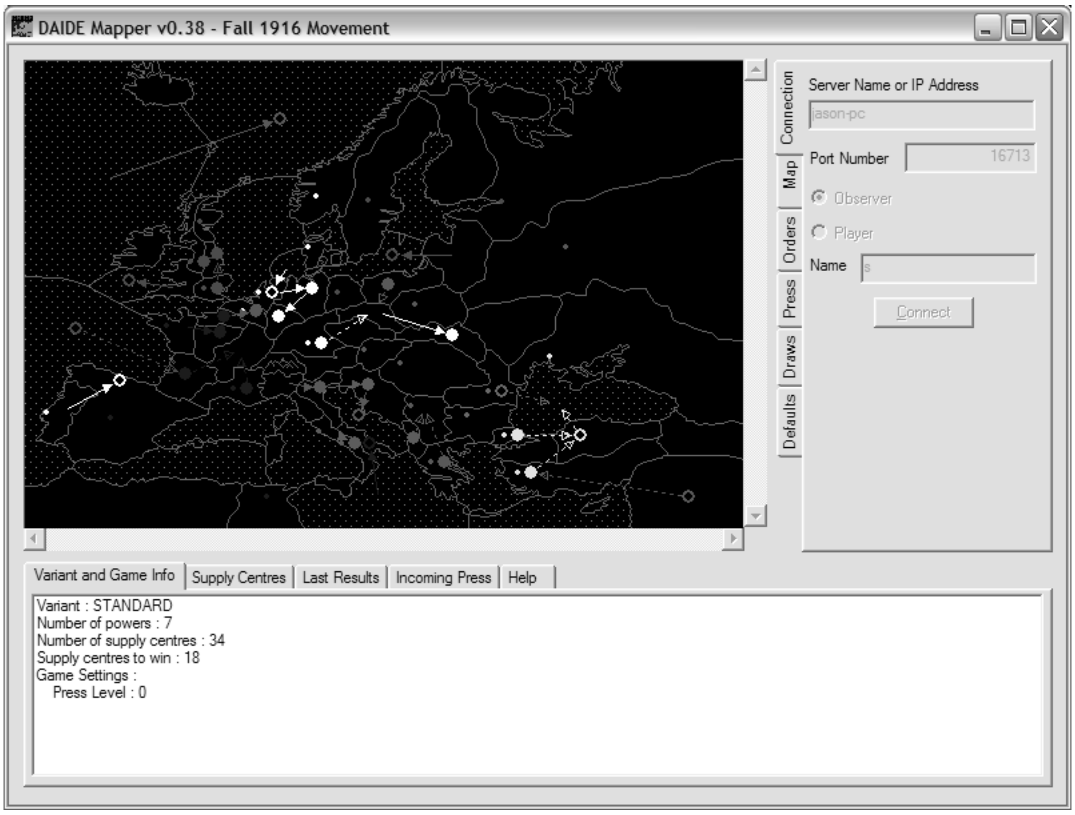
2.2.3 DAIDE Mapper

The DAIDE mapper application [15] is also written by David Norman and it allows human players to connect to the server. It also allows people to observe a match on the server. Human players are able to interact with the mapper by using its graphical interface. Users that connect via the mapper are shown the board representation and itfs current state in the graphical environment.

Figure 2 shows the mapper interface.

2.2.4 DAIDE Java AI Communication API

This is an API written by Daniel Yule [16] so that Java AI bots will be able to connect to and communicate with the DAIDE server. The framework handles the network protocols for sending and receiving messages, so that the developer does not need to deal with it and can concentrate developing



Server Name or IP Address: jason-pc
Port Number: 16713
Observer
Name: s

```
Variant : STANDARD
Number of powers : 7
Number of supply centres : 34
Supply centres to win : 18
Game Settings :
   Press Level : 0
```

Figure 2: DAIDE Mapper

the AI.

There are also many other frameworks for the DAIDE server that is written in other languages, such as David Normans C/C++ framework [14] and Fredrik Blomfs .NET APIs framework [17]. The C++ framework is also more sophisticated as it handles the map representation of the board and its current state.

However, we chose to use the Java framework to develop or bot, as even though it does not sort out the internal map representation, all the members of our group had more experience with the Java language.

2.3 Existing AIs

Due to the strong Diplomacy AI community, there are numerous existing AIs around that try to create an effective Diplomacy player. Some of which are capable of negotiating, but most bots are only capable of playing games on a strategic level only. They all offer a wide range of different approaches to the problem. Some of them use the DAIDE server whilst other's do not. The following six bots were amongst the most helpful towards our project.

2.3.1 DumbBot

Created by David Norman [20] , it is a simple but effective bot that is known for being able to beat beginner and naive players. It uses the DAIDE server to play its games and it is a popular bot amongst the DipAI community for its simplicity. It does not handle press games and so only provides a heuristic for the strategies and tactics.

This bot uses a simple algorithm that firstly assigns a value to each node in the map. This number takes into account factors such as who owns it and how strong they are, how defenceless it is, how much competition it has and how what the chances are that it will be able to successfully move there. It also uses an algorithm that averages the board so that one node's value will affect its adjacent nodes, so that it can give a more general view of the board.

Once every node has a value, it then performs a very simple algorithm to compose a move set for all its units. This is based on randomly selecting a unit and moving to the node with the highest value. Supports are performed when there are clashes and convoys are not supported.

The obvious strengths to the way that DumbBot calculates moves is its simplicity. It has an effective heuristic that calculates values for each node, and is able to produce simple move sets. However, it is far from perfect as the order sets that are generated are not effectively chosen upon. As it uses non-determinism to decide the order of units, it does not produce the best possible set. It is also not well documented.

2.3.2 Diplomat

Jaspreet Shaheed's Diplomat[18] is an effective negotiating bot that uses deceit to win. It also uses the DAIDE server to play Diplomacy games and uses in depth but efficient algorithms to find the best order sets. It can negotiate to a high press level, as well as use deception to win games. It also does not use convoys.

It first has uses a tactical component that is based on grouping its units into clusters. It first considers which provinces that it would like to occupy and then tries to move towards them. These provinces will lie on the front of the Diplomat's territory. It uses clustering by making units form small groups and then selects the best sub-plan for each group. It then joins all the sub-plans up to create a whole order set. This is done so that it does not perform too many intense calculations, as there are exponentially many possible permutations of sets of orders for each turn for a bot to decide. It then uses the clustering to try to find a trade-off between losing good plans and computing too many. It then tries to perform a static evaluation to predict which moves will succeed.

It then uses a Market Based approach for negotiation. It uses an economic based view of Diplomacy to trade resources (units) to focus on the results of the exchanges. This is also much more efficient and simpler to implement then an argumentation based implementation. It exchanges 'IOU's for supports to different powers, so that the agreement will be returned in the future. It communicates with all the other powers by setting up a simulated 'meeting' with those that are involved in the deal and treats it as an auction, where the winner is the one that makes the deal in the end. It then uses a strategy to help it on how to make use of negotiation.

The bot also included deceit to decide when to keep and break agreements that have been set. It also deals with how it will react when agreements have been broken by other players. To determine whether it should keep an agreement, it holds the notion of reliability (the chance that they will keep the next agreement that they make) and honesty (whether they are weighted towards keeping or breaking an agreement) for each power. It reacts to stabs by changing power's reliability and also considers in future how much their 'IOU' is worth to it.

While the Diplomat is an effective bot, it is very in-depth and it does not offer the best heuristic for tactical evaluation. Despite it trying to find a good general plan, it performs worse then DumbBot which does not generate the best plan [18] . However, the negotiating aspects and especially the deceit aspects of the Diplomat are effective and show an interesting take on how it can be modelled.

2.3.3 BlabBot

This is a level 20 press bot created by John Newbury [22]. It also runs under the DAIDE framework to a good standard and offers negotiation. It has built upon the DumbBot heuristics that evaluate its moves.

BlabBot uses simple negotiating techniques to create an effective bot. It works by sending peace messages to all the powers. It then uses weights in the DumbBot algorithm so that friendly supply centres and units are taken a lot less into account. If all players agrees and so it is then at peace with all players, it will perform in two fashions depending on the policies selected. It will either ignores all friendship and performs exactly like DumbBot, or it will send out a draw message every turn to try to resolve a draw outcome.

This is a very simple implementation of an automated negotiating bot. It uses DumbBot's heuristics so that it can work more towards negotiating. It then tries to form bonds with all accepting peace powers, which has an obvious advantage against no press bots, as they will be able to 'team-up' against them. When running multiples of this bot on the same game, it is therefore much more likely to succeed as it will naturally team up to battle against the other powers. However, it has no sense of deceit. It makes no attempt to exploit a friend or attempt to detect a lie. Most importantly, when faced with a lying bot, it does not change its opinion of them at all, even if they are attacked by them. However, it shows that a good bot can none the less be produced by working from another bot's tactical heuristics.

2.3.4 The Israeli Diplomat

The Israeli Diplomat created by S. Kraus, D. Lehmann and E. Ephrati [23]is a highly sophisticated Diplomacy player that uses a rich logic based language to attempts to mimic the negotiation in Diplomacy. It does not use the

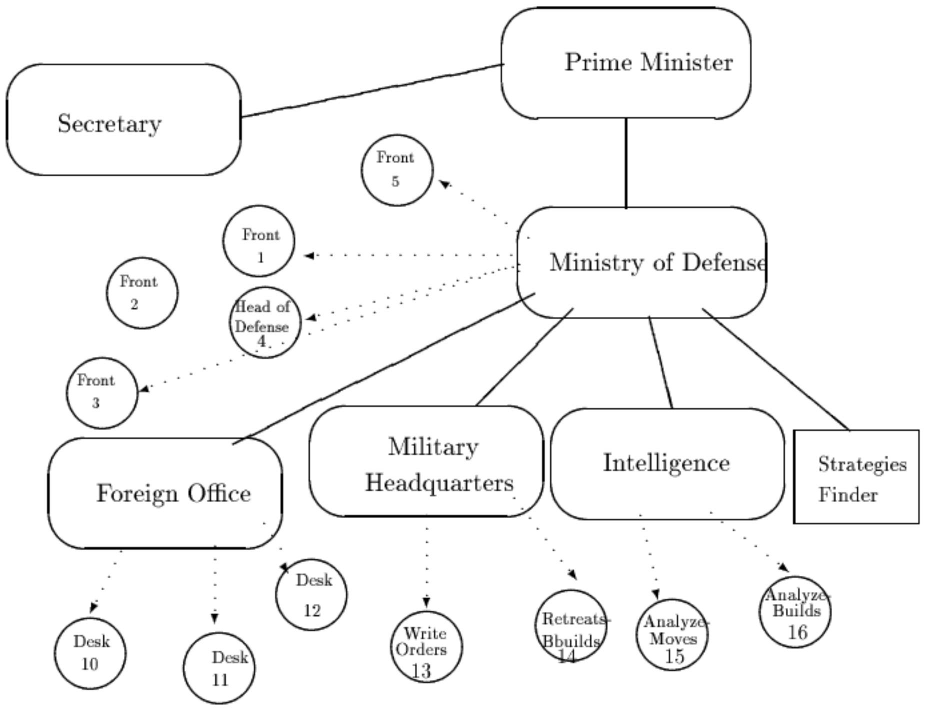


Figure 3: Israeli Diplomat Structure

DAIDE server for playing its games. It uses a multi-agent architecture that is based on real life war-time government structures. This was developed so that it can split up how each part is determined and to split each of the bot's tasks. Opponents only talk to the 'Prime Minister', whose job is to decide on the diplomats play style. If needed, the Prime Minister will pass the suggested negotiation to another role to gain and opinion before replying to the other power. Diagram 3 shows how the Israeli Diplomat is structured.

This complex architecture allows it to work at very high press levels and has been very successful in the past. Unfortunately, the hardware for which this Diplomat was designed no longer exists and due to its complex nature no one has yet to take on the task of re-programming it. So not a lot is known about the in depth nature of this architecture. There is also no detailed information regarding its heuristics or how the bot comes to form its decisions. However, the architecture of the bot provides some very interesting ideas.

2.3.5 The Bordeaux Diplomat

The Bordeaux Diplomat [24]was created by Daniel Loeb and Michael Hall and was created to explore the tactical and strategic parts of the game. It works on the principle that your enemies will always choose their best moves. Using this, the bot tries predicting its opponent's moves and decides on the best moves for its pieces given the predictions.

However, the map is too big and there are too many moves to think about in the search space. So, to overcome this, the Bordeaux diplomat breaks the map down into smaller sections, groups of provinces which make up the map. By only looking at these small groups of provinces at a time, the search space is significantly reduced. However this does make the bot slightly short-sighted when making its decisions. This bot also uses the idea of using a sphere of influence, in which a power uses its home supply centres as a point of origin and tries to expand from there. It then builds a wall between its home centres and other powers using its units, expanding the sphere as it captures more supply centres. This is illustrated in Diagram 4 .

Although it performs well and offers interesting tactical heuristics, it is not obvious how it can be adapted to include negotiation.

2.3.6 LA Diplomat

Created by A. Shapiro, G. Fuchs, and R. Levinson, The LA Diplomat [25] also does not use negotiation but instead focuses on tactical evaluation. Rather

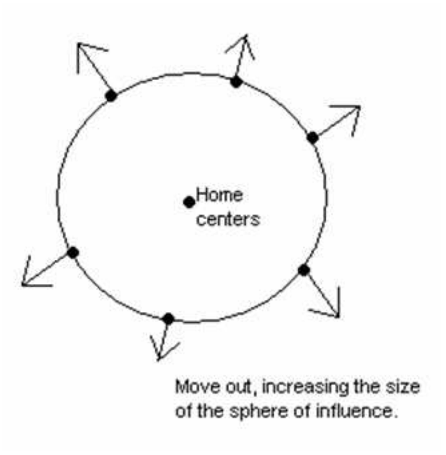


Figure 4: Sphere Of Influence

then working out which order sets is the best possible, it uses a pattern weight system and uses temporal difference learning for it to remember which moves are the best.

For each turn, the diplomat generates all the legal orders for each individual units, generates all legal order combination, matches the orders against a database and chooses the top rated moves.

This allows the bot to learn important tactics in Diplomacy. However, it was developed very slowly as it takes time for the database to build up before it became effective. The size of the database will also become an issue as it has to store a lot of data. It was also not immediately clear how negotiation could be added on to it, if at all.

3 Board Representation

3.1 Addressing Provinces Correctly

As mentioned before provinces can hold only one unit at a time, either an army, or a fleet. So it would make sense to store the list of adjacent provinces for either of these, inside an object of type Province. However, problems arise on coastal provinces, where armies and fleets can both go to. The problem is that provinces will have a weight (a number indicative of a nearby units desire to go there), and that weight will be different for armies and fleets. For example, if an unowned supply centre was one province inland of a coastal province, due to averaging, that coastal province may have a very high weight. However, if a fleet were to travel to the coastal province because of this, it would just waste its moves, because it would have no way of gaining (or even aiding) an army's attempt at) control of the supply centre. There is secondly the problem of a province with 2 coasts. A bi-coastal province would have an army occupying its entire region, however, if a fleet were to move to it, it would only be able to move to a coast of the region (e.g. North). This complicated things further, as different coasts would have different lists of adjacent provinces for fleets, but they would be the same for armies. Also, if a fleet was adjacent to a bi-coastal province, it would not know which coast of the province it would be able to move into, just that it could move into that province. Which made the weighting algorithm seemingly impossible to cope with, as the same problem arose again, which was that, a weight on one coast may not be as high for a weight on another coast, so moving to the province may be useless.

To overcome this problem we created a Node class, each Province class would contain 3 instances of type Node. Type node contains an adjacency list, and a weighting (organised via a hash table of powers, so that later we could check the value of another powers move compared to how much we value it). It also has references to a Unit and Province, as well as functions to determine its weight. Within the province class, the 3 nodes would be referenced as follows:

- Node 0 would only ever be used by armies it would be named as the province name, with the suffix of AMY.
- Node 1 would be used by all fleets which are at sea, and they would be named as the province name with the suffix FLT. Or, if it was a bi-coastal province, the name would be the province name, with the first coast suffixed.
- Node 2 would always be null unless it was a bi-coastal province, and in which case it would be named as the province name with the second coast suffixed.

On coastal provinces Node 0 and Node 1 would be used, on bi-coastal, all would be used; otherwise, only fleet and army nodes would be used for sea and inland provinces, respectively.

3.2 Units and Their Owners

Unit is created as an abstract class, which contained general methods governing standard unit behaviour. Such as what their planned action was, their location, and their controlling power. Army and Fleet are implementations of this. Their differences are just in the make up of their orders. For example, armies never got to coasts, they always go to the province, yet they have to recognise when they're supporting a unit (fleet) which is moving to the coast, so they have to obtain and parse the correct node. A class Power exists, and at game set-up time, 7 instances of it are instantiated in the one instantiation of class Map. The power class contains information on how many supply centres the power owns, references to all its units and their locations, whether it has accepted peace with us, whether it is capable of any press level greater than 0, whether we consider it an enemy, whether we want to back-stab it, and information on the banker paradigm (§7.2.1). The methods revolve around manipulating and storing these values. After each turn allowing movement, the DAIDE server sends to all the clients several messages starting with ORD. These show all the moves that had been made (or had been attempted, but cut or bounced). This is where The Diplominator learns who its actual allies are and who it's enemies are. Before the ORD phase, the map has not been updated, and so the map can calculate the possible chances another power has had to attack us. That is to say, all the instances in which they were next to us. These numbers are added up for 1 turn, which are then accumulated over several past turns. The accumulation over past turns is done so that our bot has a memory over who is friend or foe. It only records this however for every 15 turns, as alliances change. We experimented briefly with this number, and found 15 to be optimal. When all the ORD phase messages have been received, we see who actually did attack us, and that number accumulates inside the power in the same way as the chances did. With this information, we can evaluate the enemy factor. It is simply determined by this equation:

$$EnemyFactor = \frac{attacks}{chances} \text{ or if chances} = 0 \; EnemyFactor = attacks$$

It is a number used in calculating whether we accept many agreements, and how we treat the power in question, on the map. I.e. whether we feel it is appropriate to back stab said power, or to carry on treating them as allies.

The boolean for a power that's accepted peace is re-evaluated every time the enemy factor is evaluated, to make sure it doesn't exceed a set threshold.

In the Map class, all provinces, nodes, powers and units are all stored in a hash-table, with reference as a string of their name.

4 Tactics And Strategies

We decided that the best way to structure our tactical algorithm was to firstly assign some sort of values to each node, for all the provinces, based on how much we value that node. From there, we can do an evaluation for all our units to decide the best move set. As DumbBot [20] uses a similar structure very effectively, we decided to use it as a base to expand upon.

DumbBot algorithm works by firstly giving each node a value for how much we would want a unit there. As the map separates army nodes and fleet nodes, this works well for determining the difference between how much each type of unit values a province. It then generates moves for each of it's units by selecting a unit at random and then by looking at the value of each node that it can move into.

Even though it is a non-negotiating bot and uses very simple heuristics for it's tactical analysis, as well as not supporting convoy moves (which is an important aspect in the game), it is none the less still highly effective. It's simplicity and effectiveness were the main contributions on deciding to use it as a basis rather then working out a new heuristic from scratch. We also believed that it will allow us to spend more time and resources on developing a sophisticated negotiating agent, the important aspect of the game and project.

It was also pointed out in previous research papers such as ('Creating a

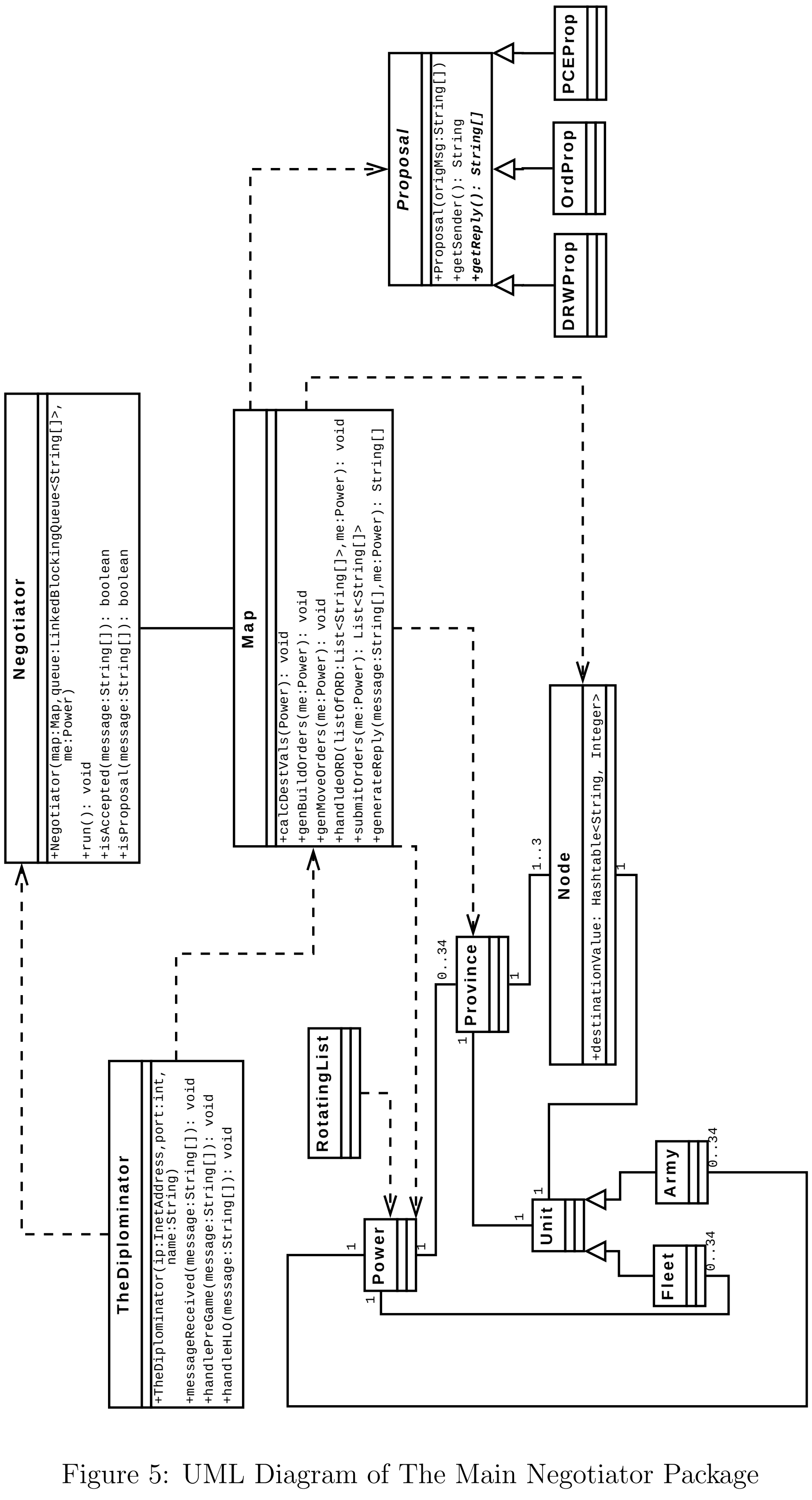

Negotiator
+Negotiator(map:Map,queue:LinkedBlockingQueue<String[]>, me:Power)
+run(): void
+isAccepted(message:String[]): boolean
+isProposal(message:String[]): boolean
Map
+calcDestVals(Power): void
+genBuildOrders(me:Power): void
+genMoveOrders(me:Power): void
+handleORD(listOfORD:List<String[]>,me:Power): void
+submitOrders(me:Power): List<String[]>
+generateReply(message:String[],me:Power): String[]
Proposal
+Proposal(origMsg:String[])
+getSender(): String
+getReply(): String[]
PCEProp
OrdProp
DRWProp
Node
+destinationValue: Hashtable<String, Integer>
TheDiplominator
+TheDiplominator(ip:InetAddress,port:int, name:String)
+messageReceived(message:String[]): void
+handlePreGame(message:String[]): void
+handleHLO(message:String[]): void
RotatingList
Province
Power
Unit
Army
Fleet
1..3
1
0..34
1
1
1
1
1
1
1
0..34
0..34

Figure 5: UML Diagram of The Main Negotiator Package

Diplobot' by Jaspreet Shaheed [18]) that a complex algorithm in determining tactics and strategies by taking into consideration different orders and plans may not necessarily lead to a better bot. An example of this is how the Diplomat bot uses intense planning and a clustering technique, but generally performs worse against DumbBot.

We also decided not to take more then one turn into account. This is because over a single turn in the game of Diplomacy, the board can change so drastically that it is too complex and computationally intensive for it to be viable. However, a blurring algorithm will be used on values which averages out and spreads the values around the board. This will be an indirect form of taking more than one turn into account, as units will move towards a 'goal'.

Our approach to the problem will also be developed in an object oriented manner in Java. We will then be taking DumbBot's heuristics as a basis to work and modify it into our own heuristics and implementing it into our own object orientated structure. This will then be used to add negotiation on top of it to create an effective automated negotiating bot.

4.1 Weighting algorithm

4.1.1 Working out the strength of a power

The strength of a power is worked out by using the following quadratic equation:

$$Ax^2 + Bx + C$$

where x is the number of supply centres that the power owns and A, B and C are constant weights.

Unlike how DumbBot works, a neutral power is defined that owns all unowned supply centres. This is so that the neutral supply centres will also have an important value assigned to it.

4.1.2 Working out the initial values

To work out a value for each node, the algorithm firstly iterates through each province and works and an initial preliminary value based on whether it is a supply centre that we own or not.

If it is one that we own then it works out a defence value, which is the calculated strength of the largest adjacent power to that province. If it is not a supply centre that we own, then it is the strength of the owning power. When it is a neutral supply centre then value will be the 'strength' of the neutral power. All other provinces have no initial value.

The idea of this initial working out is that it gives values to supply centres on how valuable they are, with supply centres that are owned by stronger enemies more important. This will make the bot more aggressive towards enemies which are stronger. This is good as it will attack those which are more dangerous (and so more likely to win), however problems may arise when the largest enemy is far away, which once the values are spread out, it will cause the bot to unnecessarily go towards them. Another advantage to this is that it will place no value on our supply centres which are not under threat, which means that our bot will not unnecessarily move units to them.

The figure 6 shows the effect of this initial working out on provinces. The map is represented by the rectangular grid and the values for each supply centre is shown via the height of each rectangular square bar.

4.1.3 Blurring the values around the map

Once we have calculated the initial values for each province, we need to then blur the values across the map so that it provides a sense of motivation, goal

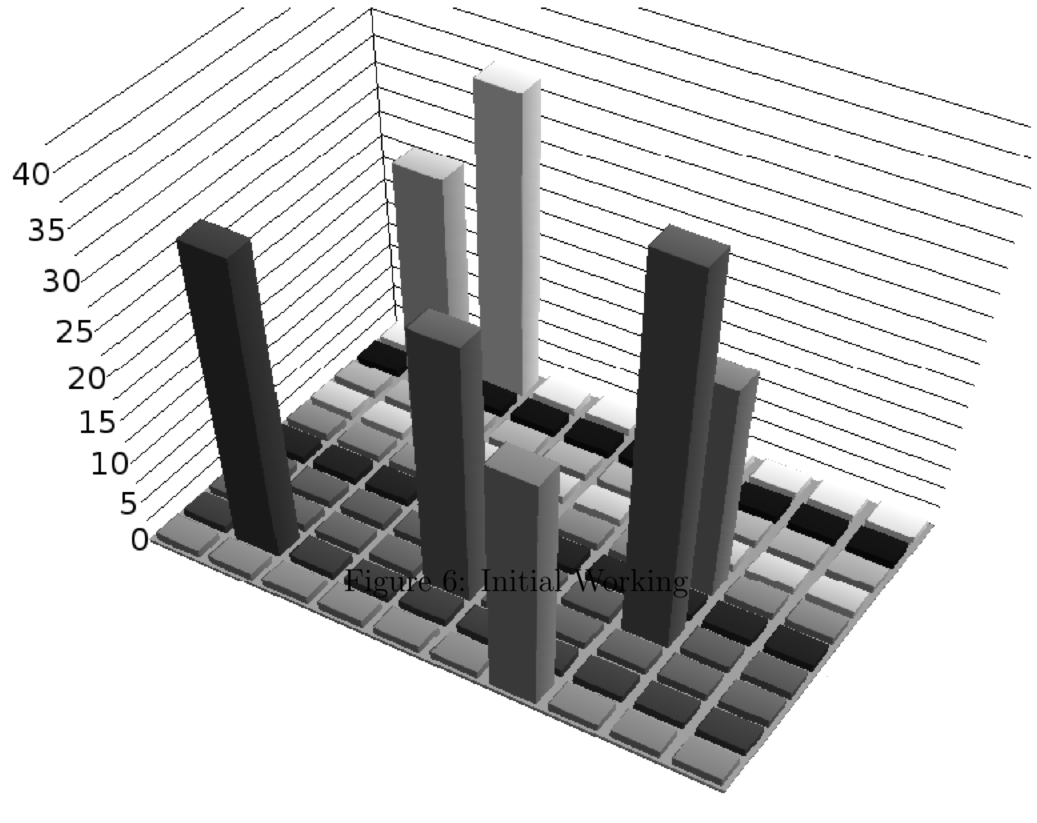


Figure 6: Initial Working

and direction on the playing board.

A board without the values blurred across the map would only provide units with provinces that have immediate values and so they would not be able to see the more general status of the board. It would also mean that if there were no blurring it would not provide a unit with as much motivation to move away from a supply centre once it has captured it, as provinces further then one province away would not be taken into account. Also, provinces which are not a supply centre but is adjacent to many supply centres would have no value, even though they would be worth a lot in the game. For these reasons, a blurring algorithm to average the values out, is applied to the map.

The blurring algorithm will be done similarly to DumbBot's algorithm where it will iterate multiple times through the entire map, creating numerous values for each node, for each province, depending what iteration it is on, and the previous iteration values. With every iteration through the map, the new value that is assigned to the node is based on the previous iteration's value and it's adjacent nodes' previous iteration's value, multiplied by a weighting value. The final destination value equates to the sum of the values for a single node, expressed in the following equation:

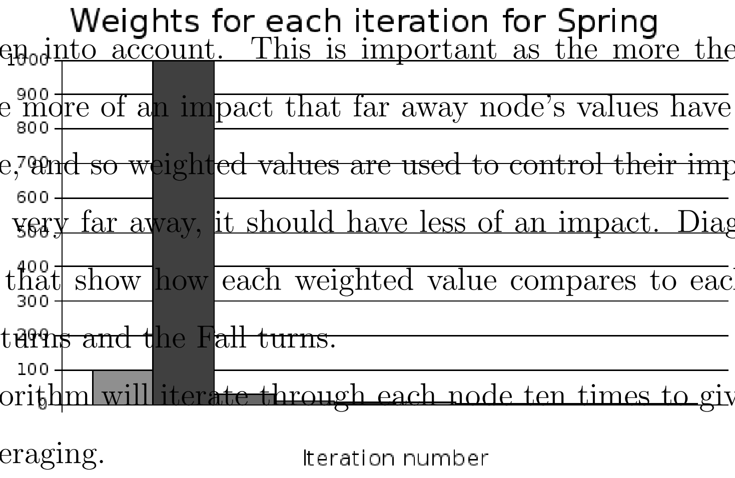
$$\sum_{n=0}^{n=10} previousValue(n) \times k_n$$

The following equation shows how each of the ten prevalues are calculated for each node:

previousValue[n] = sum(previousValue[n-1] for all adjacent nodes × previousValue[n-1] for this node)

The weighting value is applied so that the importance of each iteration

Figure 7: Spring Weight



can be taken into account. This is important as the more the algorithm iterates, the more of an impact that far away node's values have on the the chosen node, and so weighted values are used to control their importance. If a node was very far away, it should have less of an impact. Diagram 7 & 8 are graphs that show how each weighted value compares to each other for the Spring turns and the Fall turns.

Our algorithm will iterate through each node ten times to give the most effective averaging.

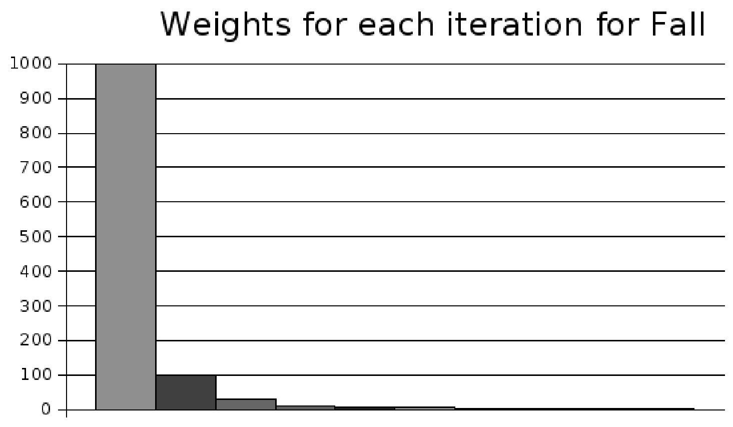
Diagram 10 shows how the averaging effects an another rectangular map.

4.1.4 Working out the strength and competition values:

For each province, we work out the strength and competition values, where the strength is the number of our units that can move into it (our strength), and the competition is the maximum number of units that can move into it that belong to a single power (our competition).

These values are for effecting the final destination value, so that we can

Figure 8: Fall Weight



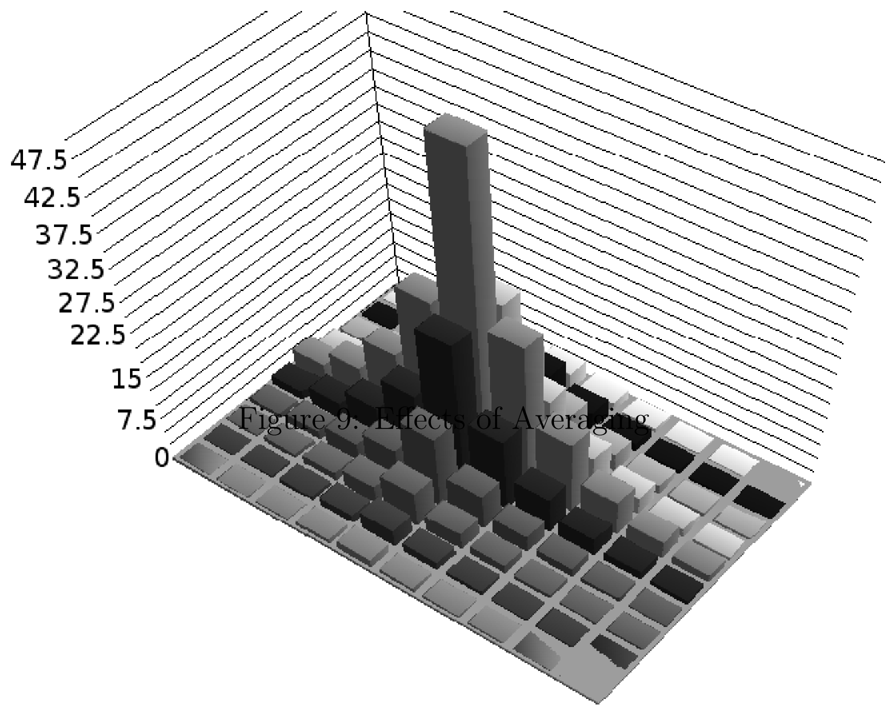


Figure 9: Effects of Averaging

determine if we are likely to succeed in moving to a province or not. If we have a single unit next to a valuable province, but that province is surrounded by enemy units, then we should consider it less, as we are unlikely to successfully obtain it.

Although this makes multiple assumptions: Enemies will not work together. We will not ask for help from another ally. Enemies want a province as much as you do (i.e. we assume that because an enemy has a lot of units next to a province, they will move into it).

However, as this is only the tactics and does not yet include negotiation aspects, the first two assumptions can be solved through the use of negotiation and modifying the competition, strength or final destination values.

4.1.5 Working out the final destination values:

This is finally worked out for each node by using the destination value worked out from the blurring algorithm and adding the strength value multiplied by a weighting and subtracting the competition value multiplied by a weighting.

This value will be the main value used for the rest of the heuristics, it represents that node's value of obtaining it for that turn.

4.2 Generating orders

4.2.1 Spring/Fall turns

To generate orders, it first selects a unit at random, and then selects the node with the highest destination value that it can move to. Like DumbBot, we have chosen to do it so that there is a random chance that it will instead select the second best or the third best, depending on how large the differences of their destination values are. This is done as part of it's non-determinism, however, to a less extreme effect that DumbBot does it.

With it's chosen node, it checks:

- If it is where it currently is already, if so then it holds.
- If it already has one of our occupying units that is not moving, then it either supports it or selects another node (depending on if it thinks that the holding unit will succeed or not).
- If it has one of our units that is already moving there, then it similarly supports it or selects another node.
- Else it moves there.

It then selects the next unit at random, which doesn't have an order yet and applies the algorithm to that unit.

Once all the units have been given an order, it can then check for wasted holds. As holds are equivalently the same to a support, then a holding unit can instead (if it is able to) support another unit, whilst still providing the same functionality. This final part of the algorithm then finds all the holding units, and checks to see if it can perform a better move. If not, then it carries on holding.

Order generation in Diplomacy can be complex and computationally intense. This algorithm is very simple and it will usually select a good move set. It will not perform any obviously wasteful moves, and it will go after the nodes which are most important to it. It briefly takes into account other powers and their units and performs supports if it needs to. It also never causes bounces to occur within it's own units and it will also never tell a unit to do something which will be an illegal move (such as making a fleet move inland, or an army into the sea).

However, it does not try to determine the best selection of orders, but instead non-deterministically selects units. This is the biggest flaw in the algorithm as it means that it will not make the best selection. Although, this has been done with other bots (such as Albert [21]) and it can be very computationally intensive. It is also not clear how we can determine whether we should use other powers and ask them to perform moves. It also notable does not perform convoys, but we believe from playing the game that it is not vitally integral to the winning of the game.

4.2.2 Summer/Autumn turns (retreat/disband)

Similarly to the spring and fall turns, it selects a random unit which it has to retreat/disband and then selects a node with the highest destination value that it can move to. If there are no nodes that it can move to, then it is forced to disband.

This is a good way to do it as it finds the best places for each individual disbanding unit. However, it does not find the most optimal positions.

4.2.3 Winter turns (builds/disband)

This is also calculated similarly to the spring and fall turns. If it has builds then it builds on the home nodes which have the highest destination value, and if it has to disband units, then it selects the unit on the node with the smallest destination value.

This is again good as it will build in the places which needs a unit most, and it disbands units on places where we need it least. It also automatically selects which type of unit to build, based on which node will have the highest value.

5 Negotiation

With an implementation of a proven tactical algorithm in place we moved on to implementing negotiation. This chapter discusses the structure of our implementation of this aspect of our AI.

5.1 Message Listener and Thread Interaction

The main class of our bot with assistance from the Java AI Communication AI acts as a message listener. When we identify that we have received a diplomatic message this is added to a synchronised queue provided by the Java standard library. When the server indicates that a turn has started we create a new thread (called Negotiatior) that runs concurrently with the message listener to process the queue. This thread then performs all the necessary work to negotiate and finally generates our moves before terminating thus ending the turn. Importantly to prevent concurrency issues arising the message listener will never modify any program data other than message queue (which is synhronised) whilst the Negotiator thread is alive preventing concurrency issues from arising.

5.2 The Negotiator Thread

The overall control flow of the Negotiator thread is illustrated in the flow chart below. After performing initialisation we process the message queue until it is empty (it may be empty to begin with). We have now reached a gap in the 'conversation' so we send our outgoing messages. We now wait for replies on the message queue for a while (a constant time set in the code which can be tweaked) waking up if we receive one. We then start this process again sending outgoing messages, processing the queues and entering the wait period. Once the wait period has timed out (i.e. negotiation has died down) we exit this loop, submit our orders and terminate. This provides a general structure for negotiation which can be customised by modifying the functions to generate outgoing orders, handling an incoming message and generating our orders once the loop has finished.

5.3 Our Negotiation Strategy

5.3.1 Peace To All

Our bot's negotiation strategy was at first very complex, as described in 7.2.1. However due to lack of time we only managed to implement part of this strategy. We correctly implemented the banking protocol, where each power has their own Credit Limit, Balance, Interest & Credit Rating. However, as of now most of that information is fairly useless because the Diplominator never discovers what would be the best time to ask for a favour from another bot, so no power can ever regain its balance to above zero, and thus will always briefly take advantage of the Diplominator, then the power the Diplominator controls would cease to be allies with the asking power. However, the negotiation part which remains fully implemented is that dealing with peace, and back-stabbing those who we are allied with when is most opportune for us. We deal with this by offering peace to all powers, to first establish which players are capable of negotiation, and also which of those which can want to be allies with us. We discovered the "peace" request in DAIDE syntax to be extremely vague; and after asking around the Official Yahoo NewsGroup for the DAIDE Diplomacy Server, everyone on there also agreed it was vague [26]. So we set it so that whenever peace was issued to another power, and they accepted, or if they issued the command

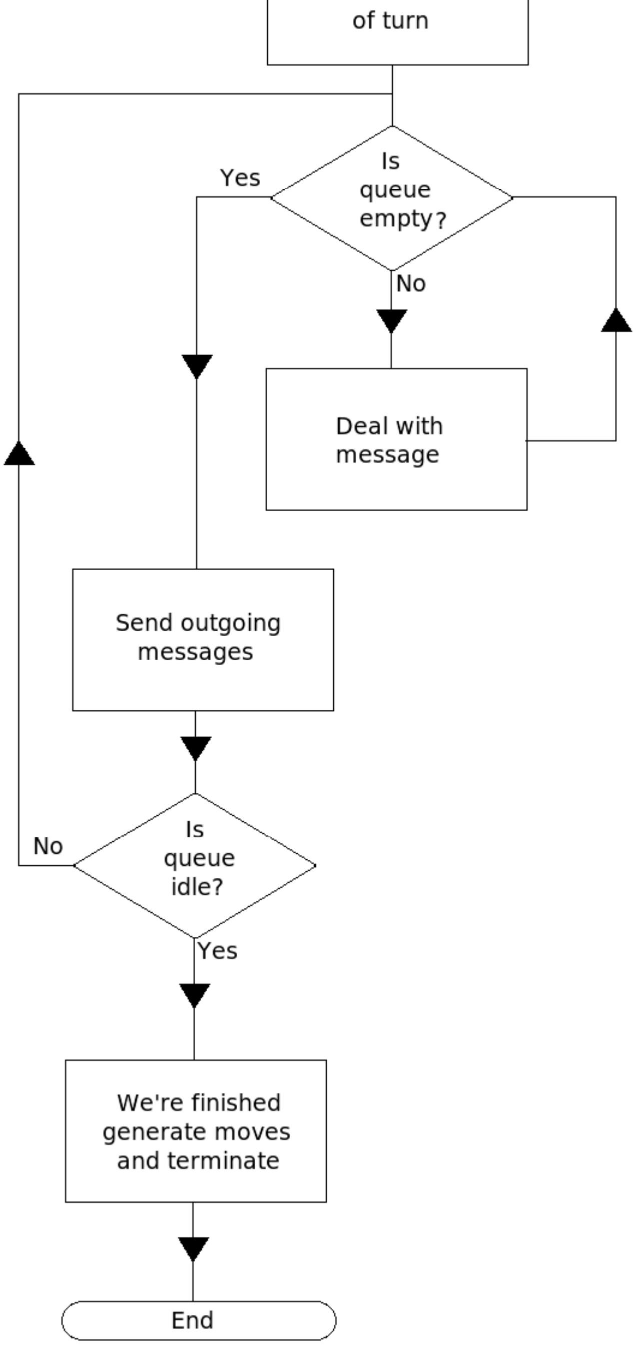


Figure 10: Control Flow For The Negotiator Class

to us (by default we would reply with yes, proividing we did not want to back-stab them at the time, nor was there enemy factor above the threshold) we would decrease the absolue weights (and thus later the average) of their home supply centres. This way, we would attack them less, generally, if they were in their homeland, but if they were in our way to get to a supply centre we needed, then we would not treat them any different to another power on the board.

5.3.2 Back-stab

This method of making peace with allies worked very well early to mid game, however, as soon as 1 power starts to dominate much more problems arose (which ultimately manifested itself as the same problem). This was basically that the powers would never break free of their general areas, they had the possibility that they would get boxed in. This happened when 2/3 powers were close together, allied and kept one in a corner, or when all of the powers were allied - none of them would ever break over the 10 power mark, or at least not for many years. The other problem was similar in that when our bot played other bots, or human players, they (if clever enough) would realise the opportune moment to disregard the friendship, and take over the power, thus making themselves a larger power on the board. This would work well for them, as clearly the pieces would not be set up to effectively guard against a border shard with an ally, so a take over could be quick. To fix this problem, we had a value of supply centres, which, when reached, the power in question disregards all friendships, and sets a boolean ”back stab” in the powers of its allies to be true. This way, instead as previously done, it would decrease the absolute value of the home supply centres of its allies, it would increase them. Thus increasing the averages around it, and making us more likely to target them. We experimented with this number and found that its optimum value lay around 10 supply centres, when we should attempt to take a lead. However we found that if we attempt to break away and take the lead, we will lose our friendships, and thus the others may all turn on us at once. If we are unsuccessful, we end up staying alone, and will probably be out shortly after. So we set it so that once we go below 9 values, after a back-stabbing attempt, we should give up on it, stop attacking our former allies, and re-request friendship.

6 Conclusion and Evaluation

In this chapter we aim to show that our bot with negotiation performs far better than not-negotiation. To do this we performed tests with our bot against DumbBot a very good no press bot.

6.1 Tests Performed and Results

Our tests consisted of pitting our bot against DumbBot in games of varying composition i.e. ranging from mostly DumbBots to mostly our bots. By observing the number of games won by one of our bots we can begin to rate our effectiveness compared to the predicted number of games we would win if the victor were simply a random player in the game. As a control we can simply repeat these tests with our bot with negotiation disabled i.e. in a press level 0 game. This should allow us to show that negotiating bots perform better than non-negotiating ones.

Composition	**Won by us**	**Total Games**	**Win % for Us**	**Expected Win %**
6 Dips, 1 Dumb	11	14	78.57%	85.71%
5 Dips, 2 Dumb	11	14	78.57%	71.43%
4 Dips, 3 Dumb	8	14	57.14%	57.14%
3 Dips, 4 Dumb	7	14	50.00%	42.86%
2 Dips, 5 Dumb	3	14	21.43%	28.57%
1 Dip, 6 Dumb	1	14	7.14%	14.29%

Table 2: Table Showing Results for the Diplominator for Various Game Compositions (without Negotiation)

Composition	**Won by us**	**Total Games**	**Win % for Us**	**Expected Win %**
6 Dips, 1 Dumb	18	18	100.00%	85.71%
5 Dips, 2 Dumb	18	20	90.00%	71.43%
4 Dips, 3 Dumb	19	20	95.00%	57.14%
3 Dips, 4 Dumb	10	20	50.00%	42.86%
2 Dips, 5 Dumb	7	20	35.00%	28.57%
1 Dip, 6 Dumb	1	20	5.00%	14.29%

Table 3: Table Showing Results for the Diplominator for Various Game Compositions (with Negotiation)

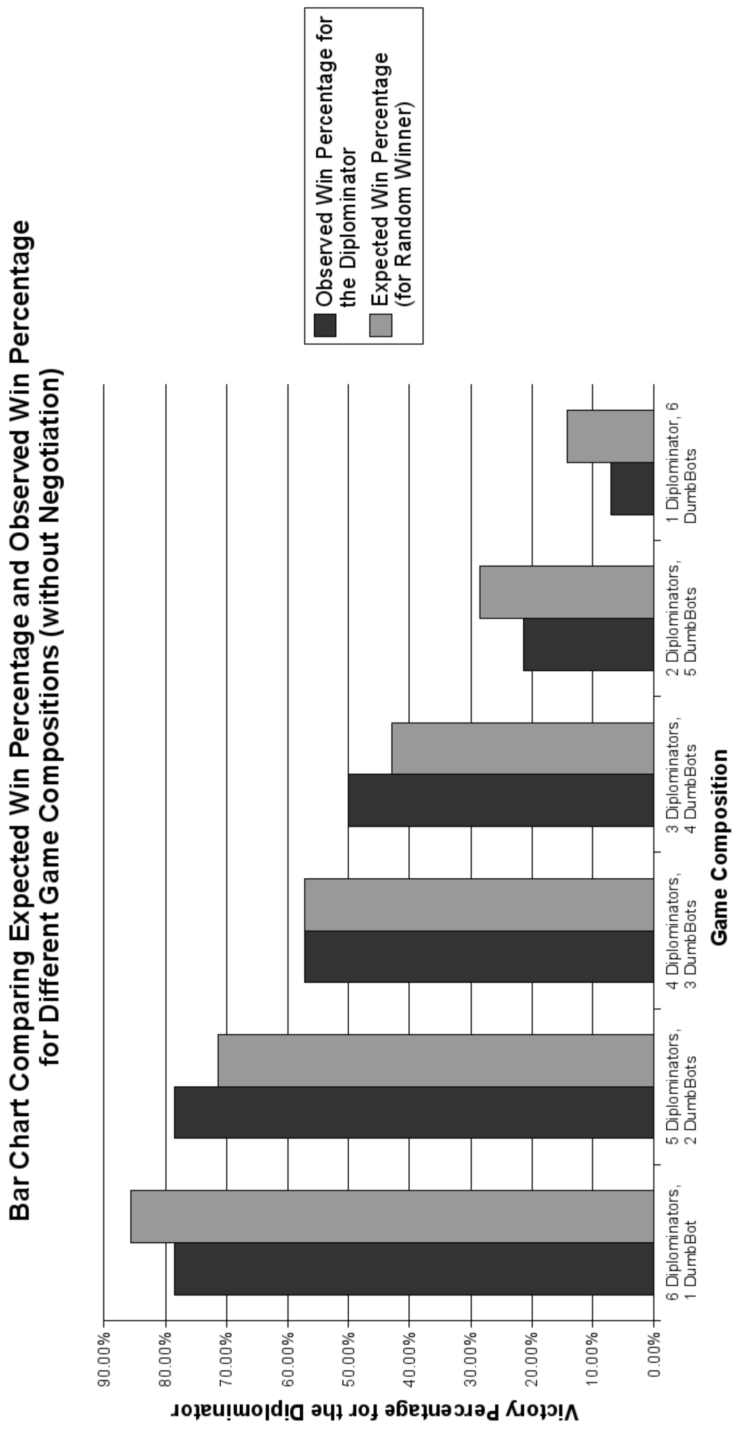


Figure 11: Showing Results for the Diplominator for Various Game Compositions (without Negotiation)

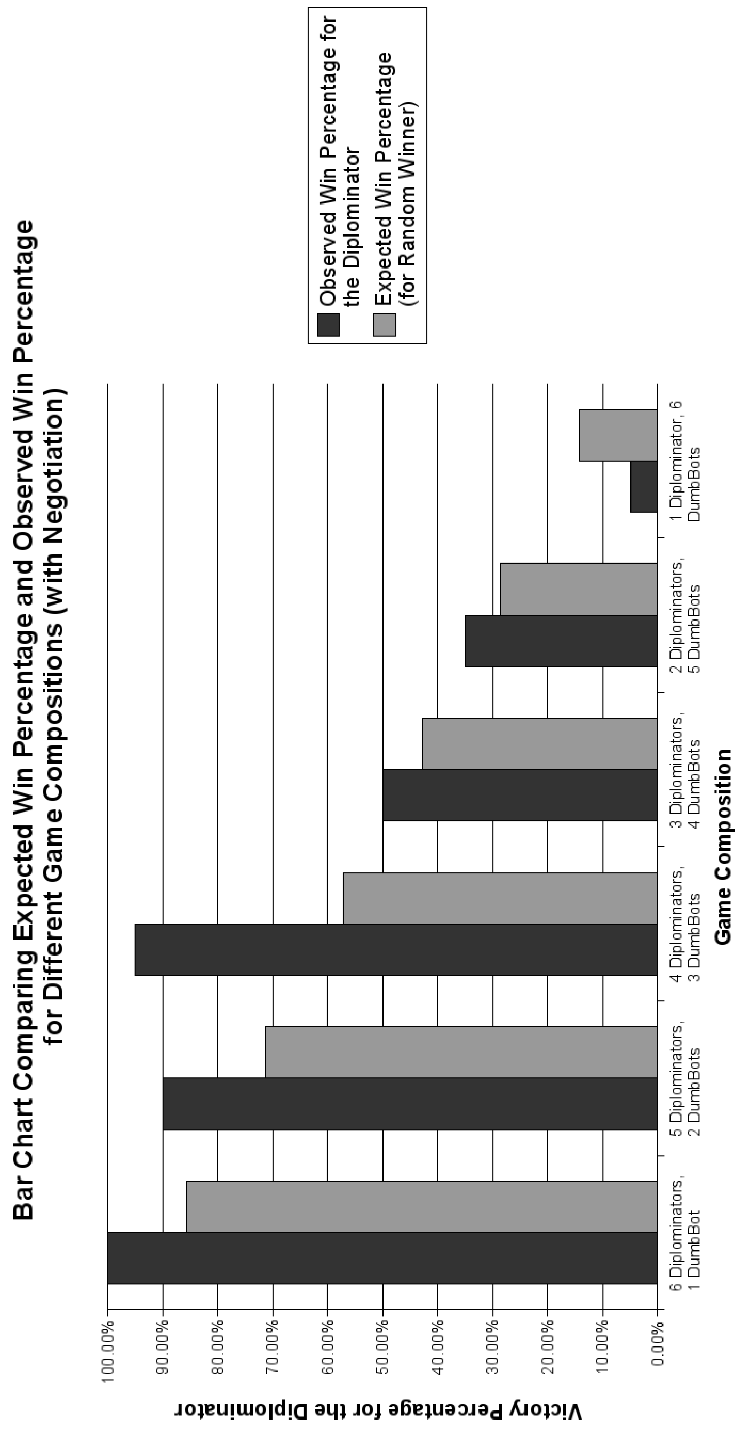


Figure 12: Showing Results for the Diplominator for Various Game Compositions (without Negotiation)

6.2 Analysis and Conclusion

The first chart shows that when our bot does not negotiate we perform quite similarly to DumbBot i.e. we beat it roughly as often as would be expected by random chance across the range of game compositions. This is as expected as without negotiation works very similarly to DumbBot. When we enable negotiation and repeat the tests the second chart shows us that we beat DumbBot across the board except in the case where there is only one of us (where negotiation is obviously going to be ineffective.

Furthermore we can also see that the more of our negotiating bots there are the better we play. This is probably due to the negotiating bots ganging up on and eliminating non-negotiating ones.

From this we can conclude that even rudimentary negotiation gives a player an advantage in Diplomacy. This is precisely what one would predict.

6.3 Evaluation

We have shown reasonably strongly that negotiation improves play in the game of Diplomacy. This conclusion could obviously be strengthened by more repetitions (time constraints limited the duration of our testing). With more time and careful analysis it should be possible to go much further. For example one could analyse which groupings of powers were particularly effective when played by negotiating bots Study into tipping points where alliances break down may also be useful.

More interesting results could also be acquired by improving our negotiating strategy as currently it is quite simple. Particularly analysis of the rewards and penalties for deceit would be of interest. We discuss further possible interesting expansions to the negotiating strategy in the following chapter(7.2).

7 Future Work

There is vast potential for future work in this area. Diplomacy AI development is still relatively young compared to more studied games. In particular we can break these potential extensions down into three areas.

7.1 Improvements to the DAIDE framework

We found ourselves frequently parsing very similar messages. As we did not produce a specific parser/compiler for the DAIDE syntax this was rather time consuming. The development of a Java parser/compiler to automate this work would allow AI developers to focus on higher level concerns rather than worry about IO issues. Of particular use may be the ANTLR tool [19] as this is specifically designed for this kind of problem.

7.2 Improvements to Our Code Base

We succesfully developed a general negotiation system for Diplomacy bots with potential for improvement by modfication of several core methods (see section 5.2). This should allow the bots strategy to be modified without changing its overall architecture. Whilst we ran out of time to use this method for generating sophisticated negotiation (we only ended up operating at DAIDE press level 10) we believe there is vast potential for extension.

7.2.1 Banker Paradigm

Specifically we intend to, but run out of time to implement a DAIDE press level 20 bot based around a banking system. Press level 20 opens up the possibility for bots to suggest moves to each other which may either be accepted or rejected. However only one move may be suggested in one offer. This results in the problem of our bot having to consider whether it is worthwhile to help another player based on their liklihood of reciprocating in the future. This problem seems to map very well to how banks decide whether or not to lend money to individuals. We believe a negotiation strategy modelled around how banks act in these circumstances may form the core of an effective press level 20 bot. This strategy would contain several key components.

Method for Evaluating Utility of Cooperation In order for a banking system to work effectively we must have a currency. In the simplest case where at press level 20 we are simply discussing the values of move requested by another player we could use DumbBots heuristic to calculate a destination value (from the perspective of the player asking) for the province they are requesting assistance in. This gives a simple heuristic for evaluating that assistance. More complicated valuation functions could be developed to create a more sophisticated bot.

Method for Evaluating the Creditworthiness of Opposition Players Obviously it is foolhardy to assist players who are unlikely to help us in return. Thus an intelligent banker should estimate how much of a credit risk each other player is. This could range from a simple consideration the percentage of times the opponent has assisted us when asked to far more sophisticated heuristics taking into account more diverse criteria.

A Credit Limit In order to prevent our banker extending infinite credit and assisting opponents indefinitely their should be some notion of a credit limit. Just as there is a limit on our overdrafts beyond which we cannot borrow more we should at some point stop assisting opponents until they pay us back. This limit could be dependent on many factors for example the closeness of our relationship with a particular power or there previous banking history.

Interest By taking into account the above factors it should be possible to charge some notional interest for our assistance. This can be used to ensure we profit from such exchanges.

We believe it would be of particular interest to analyse how changing the honesty of a banker affects it's success. Do bankers who are honest do better or is it better to renege on one's deals.

7.2.2 Other Extensions

This is but one method of improving our code. Due to it's architecture futre developers should be able to create a variety of negotiting strategies simply by modifying a few methods called by the negotiatior class. Specifically methods to generate outgoing orders, process individual incoming messages and generate moves once negotiation is finished. This abstracts away many of the issues with building a negotiating player.

7.3 Open-ended Improvents

The scope for Diplomacy AI development is huge. Specifically we believe the following work would be of interest:

Negotiation at High Press Levels Currently most bots available for the DAIDE framework negotiate only at or below press level 20. There is a large amount of room for the development of bots far more sophisticated this. The DAIDE framework also provides incremental stepping stones (i.e. gradually increasing press levels) towards this.

Tactical Play beyond DumbBot Must current bots do not play at a tactical level much better than DumbBot without time consuming algorthims for move generation. By creating better tactical algorithms we may not only create better tactical bots but we may also improve the potential for further modifying these tactics with strategic and negotiating play.

Machine Learning Currently most bots learn very little during the course of play. If a bot can be designed that interprets and learns from the actions of its oppenents a better bot may be created.

References

[1] Game publishers web site *http://www.wizards.com/default.asp?x=ah/prod/diplomacy*

[2] "Board Game Geek" article on Diplomacy *http://www.boardgamegeek.com/game/483*

[3] Diplomacy AI Development Environment (DAIDE) Homepage *http://www.daide.org.uk/index.xml*

[4] The Rules of Diplomacy *http://www.wizards.com/avalonhill/rules/diplomacy.pdf*

[5] Diplomacy Wiki *http://www.diplom.org/bin/dipwiki.pl*

[6] diplomacy-archive.com *http://www.diplomacy-archive.com/home.htm*

[7] Opening Strategy, Part I: Philosophy by Jake Orion *http://www.diplomacy-archive.com/resources/strategy/articles/opening_strategy1.htm*

[8] English Opening Strategy by Stephen Agar *http://www.diplomacy-archive.com/resources/strategy/articles/england.htm*

[9] Brainshell *http://www.brainshell.org/*

[10] Maginot *http://maginot.diplomacy.se/*

[11] Client List *http://www.daide.org.uk/clients.xml*

[12] Official DAIDE Syntax *http://www.ellought.demon.co.uk/dipai/dpp_syntax.rtf*

[13] DAIDE Server Application *http://www.ellought.demon.co.uk/dipai/aiserver.zip*

[14] C/C++ Framework *http://www.ellought.demon.co.uk/dipai/aiclient.zip*

[15] DAIDE Mapper *http://www.ellought.demon.co.uk/dipai/aimapper.zip*

[16] Java Framework *http://web.unbc.ca/ yuled/dip/*

[17] .NET Framework *http://daidedotnet.sourceforge.net/*

[18] Jaspreet Shaheed's Diplomat *http://www.doc.ic.ac.uk/ jss00/old/final_report.ps*

[19] ANTLR Parser Generator *http://www.antlr.org/*

[20] DumbBot Source Code *http://www.ellought.demon.co.uk/dipai/dumbbot.zip*

[21] Albert Source and Creator *http://ca.geocities.com/stretchy@rogers.com/Albert.htm*

[22] BlabBot source and Creator *http://uk.geocities.com/johnnewbury1/diplomacy/blabbot.htm*

[23] Israeli Diplomat source and Creators *http://www.cs.biu.ac.il/ sarit/Articles/13.ps*

[24] Bordeaux Diplomat *http://devel.diplom.org/DipPouch/Zine/list.html#DPP*

[25] The LA Diplomat Documentation *http://www.cs.ucla.edu/ ashapiro/ShapiroA_CG2002.pdf*

[26] The Official DAIDE Yahoo NewsGroup *http://games.groups.yahoo.com/group/dipai/*