Master Thesis
Computer Science
Thesis no: MCS-2004:20
August 2004

Multi-Agent Diplomacy

Tactical Planning using Cooperative Distributed Problem Solving

Fredrik Håård

Department of Software Engineering and Computer Science
Blekinge Institute of Technology
Box 520
SE - 372 25
Ronneby Sweden

This thesis is submitted to the Department of Software Engineering and Computer Science at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Computer Science. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Author: Fredrik Håård
E-mail: fredrik@haard.se

Advisor: Stefan Johansson
Department of Software Engineering and Computer Science
E-mail: stefan.johansson@bth.se

Department of
Software Engineering and Computer Science
Blekinge Institute of Technology
Box 520
SE - 372 25
Ronneby Sweden

Internet : www.bth.se/ipd
Phone : +46 457 38 50 00
Fax : + 46 457 271 25

Abstract

While there is a broad theoretic foundation for creating artificial intelligence based solutions for two-player games, such as Chess, the multi-player domain is not as well explored and artificial intelligence solutions for multi-player games is often flawed. This report is an attempt to apply a multi-agent approach to a multi-player game, and use distributed problem solving to create viable plans in an environment of huge search spaces and multiple adversaries. An automated player (bot) for the game Diplomacy was created using distributed methodologies, and tested against other existing bots. The tests show that the bot developed can outperform opposition in score while being competetive in speed.

Keywords: Diplomacy, AI, Games, Agent, Multi-Player

Acknowledgements

I have had much help from several people while writing this report. First and foremost, I would like to thank my supervisor Stefan Johansson, who has given invaluable support, feedback and ideas during the work with this report — he has helped me with everything from algorithms to LaTeX.

I would also like to thank the jDip team, especially Zach DelProposto for the creation of and help with jDip, as well as Henrik Bylund whose DAIDE library I have used. I am indebted to David Norman and François McNeil who have developed the bots I have tested against, and I wish to thank the people in the DIPAI Yahoo group for a lot of help regarding DAIDE and Diplomacy AI in general. A very special thanks goes to David Norman and Andrew Rose - without DAIDE, HaAI would have remained a figment of my imagination.

CONTENTS

1. INTRODUCTION

The creation of artificial intelligence (AI) players for two-player games is a well known domain of computer science and mathematics. For games with manageable search spaces, these methods include extensive searching for possible moves, and weighting of found positions to find the best move from the current position. Such methods can then be improved by using heuristic methods, such as Alpha-Beta pruning [RN95]. However, this solution is not applicable to all kind of games. Diplomacy presents several problems when developing an AI. Firstly, there are huge search spaces - the exact number of unique openings is 4,430,690,040,914,420 not counting useless supports [Loe95]. This clearly makes extensive searching all but impossible. Secondly, it is a multi-player game, and this make evaluating a position very hard, since the *strongest* position strategically and tactically is not always the *best* position [LH92][Loe95]. These characteristics make Diplomacy a very interesting game to study, since the uncertainty and unmanageable search spaces is something that it has in common with the real world, while the software environment and rules make it a viable domain for experiments.

1.1 Problem description

This work intends to explore the possibility of using a multi-agent architecture to create an automated Diplomacy player (bot), in an attempt to discern wether a distributed solution can successfully compete with centralized solutions in games with high complexity and huge search spaces. The domain was chosen out of personal interest and since the nature of the game makes it hard for classical approaches to handle.

1.2 Method description

The method for this study consist of the development of a Multi-Agent based bot for Diplomacy, and evaluation of the performance of that bot compared to existing bots, through the means of a Diplomacy tournament. A call for participation was sent out to the Diplomacy AI community; including the bot developed, four bots in six versions were entered into the tournament.

1.3 Software agents

Throughout this report, we will use the definition of an agent used by Russel and Norvig - an agent is any entity that perceives and acts. The agents used (Section 3.2.3) are utility-based and semi-autonomous - while they have own goals and view of the board, they are bound by a rigid protocol and managed by the game engine [RN95].

1.4 Outline

In the following chapter, Diplomacy will be described in detail and the AI environment will be introduced. Included is also a brief description of some previous approaches to Diplomacy AI. In Chapter 3, the bot developed as a part of this report is described, and in Chapter 3.3 and 3.4 the experiment setup and results are presented. Following the experiment presentation are the discussion and conclusions drawn from the experiments.

2. DIPLOMACY AND DAIDE

Diplomacy[1] [2] is played on a map resembling Europe at 1901 (Figure 2.1). Each player represents one of the Great Powers of Europe at the time - Russia (white), Turkey (yellow), Italy (green), France (light blue), England (purple-blue), Germany (grey-black) and Austria-Hungary (red)[3]. The map

[1] Diplomacy is ©Hasbro Inc.

[2] Diplomacy as described here is the "No-press Standard" variant - it is played on the original map, but without negotiations. More information on Diplomacy variants can be found in [arc04] and [pou04]

[3] The colors specified may vary - these are the colors the jDip user interface uses.

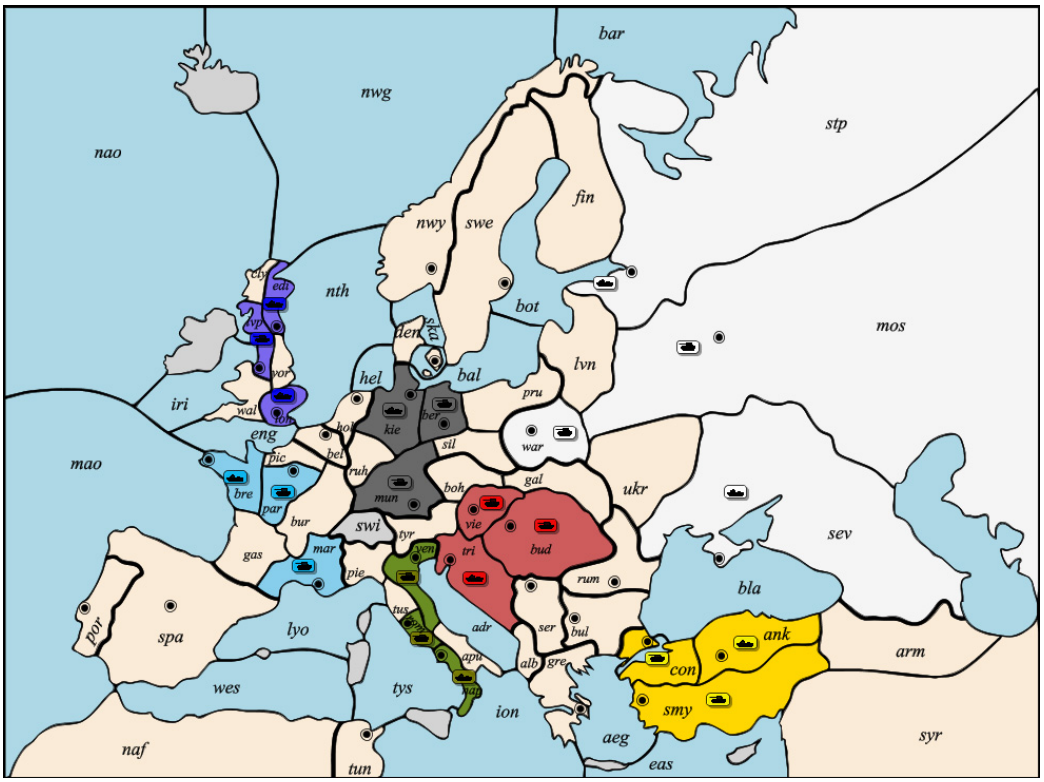

Fig. 2.1: The Diplomacy map at the beginning of the game, with province abbreviations.

is divided into *provinces*, and each province is connected to other provinces through borders. A province can be either a *land province* or a *sea province*. Additionally, there are *coastal provinces*, that is a land province that is accessible from a sea province. To further confuse things, there are also *multi-costal* provinces. These are provinces that have multiple discrete coastal areas. For example, Burgundy (bur), in the east of France, is a land province without coast (*landlocked*). Brest (bre), in the north-western part of France is a coastal province, and the English Channel (eng) is a sea province. St. Petersburg (stp) is a multi-coastal province, having a north coast (nc) in Barents sea (bar) and a south (west) coast (sc) in the Gulf of Bothnia (bot). The different types of provinces are defined in how they interact with units. There are two kinds of units, *fleets* and *armies*, denoted 'F' and 'A' respectively. Armies can only traverse land provinces and only over land borders. Armies are not affected in any way by coasts or multi-coastal provinces. Fleets on the other hand may move only in sea and coastal provinces, and may never enter a landlocked province. Furthermore, the multi-coastal provinces have more than one discrete coast, and fleets moving to a multi-coastal province must specify which (bordering) coast to enter. A fleet may not move directly from one coast to another within a multi-coastal province. Some (land) provinces are *production centers* - marked with a dot on the map. Each such production center allows its owner to construct and maintain one unit - in other words, the maximum number of units a power can control is the number of production centers he controls.

2.1 Seasons

Turns in Diplomacy are measured in *seasons*, or phases. The seasons are *spring movement, summer retreats, fall movements, winter retreats and winter adjustments* (or *winter builds*). During each movement season, all powers submit orders for their units. During retreats, units that have been dislodged are retreated or disbanded, and during the adjustment phase supply center ownership is decided and units are built or disbanded. When the game begins, all players control three centers except Russia, which controls four, and units on those control centers. These centers are considered the players *home provinces* - shaded on the map in the color of the owner. The significance of home centers is that it is only in the home centers that new units can be constructed. In addition to these 22 centers, there are 12 uncontrolled provinces at the beginning of the game. A power can take control of an uncontrolled center or a center of another player by occupying it with a unit during the winter adjustment phase. He then controls the center until an-

other player takes it over in a similar fashion. Seasons are abbreviated using [season][year][type] - the game begins in S1901M (Spring 1901 Movement).

2.2 Goal

The aim of the game is to dominate Europe by controlling at least 18 centers - the first player to do so is the winner (by a *solo* victory). Since there are positions that can result in a deadlock, the game can also be ended at a predefined stage (such as winter adjustments 1950), or when all surviving players agree on a draw. Games ended prematurely are always considered a draw between all surviving players.

2.3 Orders

A unit can in a season only move to and interact with units in provinces bordering the province they reside in. A fleet may not directly interact with units in a landlocked province, and armies may not interact with fleets in a sea province with the exception of *convoys*. Convoying is the act of moving an army over sea by fleets. To do this, the army needs to be ordered to move by convoy through several provinces (at least three: origin, convoy, and destination). The convoy must move from a coastal province, through one or more sea provinces, to a coastal province. In every sea province where the convoy moves through, there must be a fleet ordered to carry through the convoy. Any unit can also be ordered to *hold* - stand ground, or to *support* the action of another unit. In any season, orders for all units are entered secretly by each player and then revealed and carried out simultaneously.

2.4 Conflict

All units are of equal strength, and whenever two units try to enter the same province, a standoff occurs (*bounce*) and the involved units do not move. This also happens when a unit tries to enter a province where there already is another unit (that is not moving away). Since no unit can win over another by itself, *support* has to be added. A unit can give support into any province it could move to. Support is given either to hold (defend) or *move* (attack). Support is only valid if the supporting unit is not attacked itself - even if that attack is unsuccessful. The act of attacking a supporting unit is known as *cutting support*. If a supported unit moves to a province where there is an unsupported unit, the attacked unit is *dislodged*. During the following retreat phase, dislodged units must retreat to an unoccupied province other

than the one the attacker came from, or be *disbanded*. The general rule of movements is that equal strength bounces, superior strength prevails. For a more in-depth discussion of the rules and their implications, refer to the rule book [Cal00] and/or "*The Game of Diplomacy*" [Sha78].

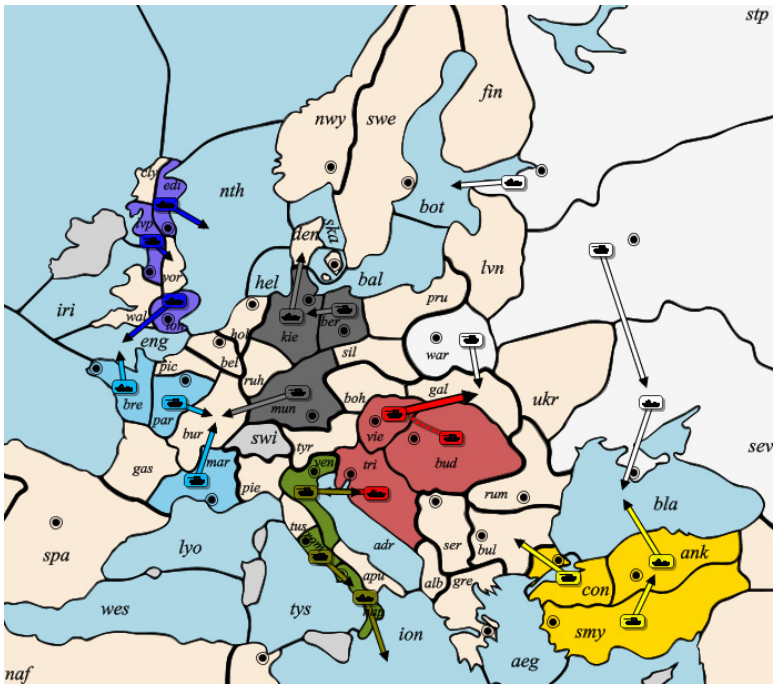

Fig. 2.2: Opening moves for a game of Diplomacy. Arrows indicate movement orders, the dashed arrow from the Austrian army in Budapest (bud) indicates that it is supporting the movement of the army in Vienna (vie) to Galicia (gal).

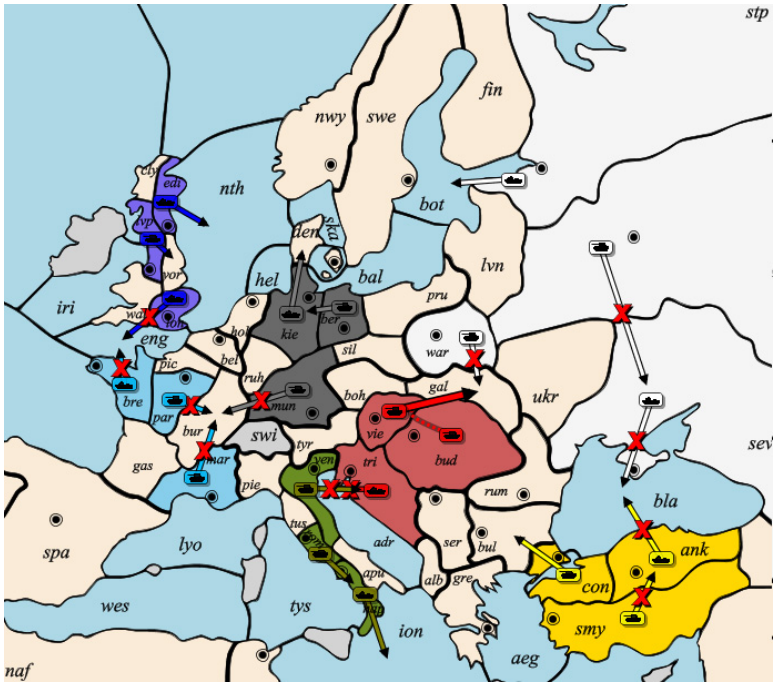

Fig. 2.3: The adjudicated results of the orders in Figure 2.2. Crossed arrows are failed orders. Note the standoff in Burgundy — France used a self bounce to protect the area without entering it.

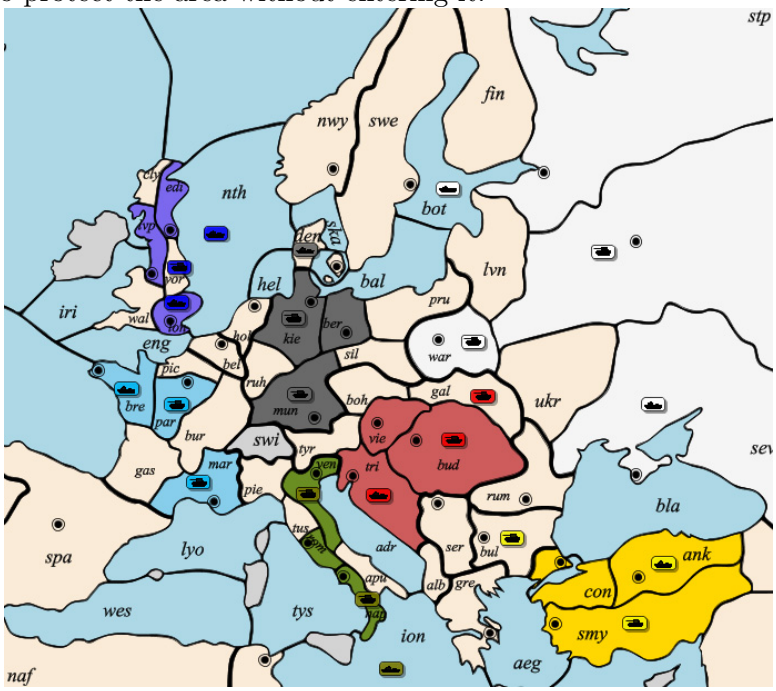

Fig. 2.4: The resulting position from Figure 2.2 and 2.3.

2.5 Previous approaches to Diplomacy AI

There have been several previous attempts to create an automated Diplomacy player, and this section will describe a few of them that have been used during the work with this thesis.

The Israeli Diplomat

One of the earliest, and best documented, attempts to create a Diplomacy bot was the Israeli Diplomat. It was primarily concerned with the diplomatic aspect of the game, and was reportedly quite successful - it played better than its human counterparts. The Israeli Diplomat uses an agent based approach, and distributes tasks between agents that are ordered in a hierarchical fashion [KL95]. Source and binaries for the Diplomat seem to be lost.

The Bordeaux Diplomat

The Bordeaux Diplomat is based on an optimized best-first searching algorithm, seeded with best-guess moves. It uses scripted "book openings" to increase performance, and an evaluation method that creates areas of varying importance that the bot should try to control. The strategic and tactical planning seems to be done through searching with heavy pruning to offset the huge search space [Loe95]. Like the Israeli Diplomat, source and binaries for the Bordeaux Diplomat could not be located for use in this report.

DumbBot

DumbBot, while from the beginning not attempting to be a serious attempt at an artificial intelligence (AI) for Diplomacy, has proven very successful. Originally written by David Norman to serve as an example for users of Diplomacy AI Development Environment (DAIDE), it uses a rather simple algorithm, that nonetheless has proven competitive and indeed beaten more serious attempts at creating Diplomacy AI:s. DumbBot works by first calculating values for all provinces, and then creating orders based on those values. When evaluating provinces, it takes into account supply centers, owner size and proximity, as well as the attack strength it has on the province. Then it tries to move units to the highest ranked province, with random chances at moving towards lower ranked provinces with the chance declining proportionally to the values computed. If the unit is already at the best place it can reach, it holds, and if another unit is already occupying the province or is moving there, it either supports the unit (if it is not already guaranteed

to succeed) or picks the second best move. Retreats, builds and disbands are handled in much the same way - try to get units from low-ranked to high-ranked provinces. [dai04]

2.5.1 RandBot

RandBot is a bot written by David Norman, and it simply creates a random set of valid moves from the moves available to each unit.

2.5.2 DiploBot v1.2

DiploBot is developed by François McNeil. DiploBot's tactical analysis is based on setting weights on all provinces and then analyzing possible routes. It first analyzes the threats around its own supply centers and units, and adjust priorities before analyzing routes. It uses a stepped-iterative approach where a sequence of different modules modify the weights of each province based on some criteria. Once every module is done, it passes the resulting weighted map to the routes analyzer which returns a sorted list of routes per unit. The sorting considers the value of the route, the ratio of threats/supports, the priority flags set by the threats analysis mentioned above. It then moves down the sorted list and tries to assign the best route for each unit. For building, it selects the empty supply center that is the most threatened and builds a unit based on the ratio of neighboring provinces that are lands or seas. For removing, it simply removes the unit that is the furthest away from the home provinces.

2.5.3 Man'chi

Man'Chi by Brian Roberts has the most complete strategic planning of the bots available [man04]. Two versions are used in our experiments:

- *AttackBot* initially picks a random neighbor and attacks that player until somebody else attacks it - then it targets the player that attacked it. Pays little attention to defense.
- *DefenseBot* - like the AttackBot but with heavy emphasis on defensive goals with only minor attacks against its target.

2.6 DAIDE

DAIDE - Diplomacy AI Development Environment - is an environment to allow automated Diplomacy players to compete against each other. It consists

of a communications model and protocol, and a language for bots to negotiate and specify instructions. A server which bots can use to play against each others is available (developed by David Norman) for bots using this environment [dai04].

3. HAAI

HaAI is the name selected for the attempt to create a multi-agent based Diplomacy player. The bot was implemented by the author in Java and is able to connect to the DAIDE server to play.

3.1 Multi-Agent Systems & Cooperative Distributed Problem Solving

A multi-agent system is a system that uses several agents to complete some task. According to Wooldridge [Woo02], multi-agent systems are suitable under a number of circumstances. The main reason for using a multi-agent approach for Diplomacy is very same as the first cited by Wooldridge - "[the environment is] highly dynamic, uncertain or complex". The multi-agent approach is also attractive since it allows for distribution of local tactical analysis, simplifying the analysis process.

Cooperative distributed problem solving - CDPS - is the use of a multi-agent system with semi-autonomous agents working together to solve a problem. The first task for any such system is the decomposition of the problem. The second stage in the process is the sub-problem solving phase, and the third and final stage is the answer synthesis, or the *plan formation* [Woo02]. There are several models for problem decomposition and plan formation, with the Contract Net [Smi77] being the one that has had most impact on the work with this thesis.

3.2 HaAI Design and Architecture

HaAI consists of a world model, a communications module, a number of 'unit agents' and an engine to run it all. The world model is based on the jDip adjudicator and world model, an open source implementation of Diplomacy [jdi04]. The world model also contains distance matrixes for fleets and armies. The communications module uses Henrik Bylund's Java DAIDE library [byl04] to communicate with the DAIDE server. The game engine is extremely simplistic - it receives updates from the communications library

and initiates the processing of the unit agent, and passes any resulting actions back to the communications module.

3.2.1 'HaAIEngine' - The bot engine

The bot engine receives events from the communications module and initiates the actions of the unit agents. It initiates the world model ('WorldModel') and keeps all unit agents in a list ('AgentList'). When actions from the unit agents are required, the engine will collect a list of goals from the agents, and then initiate plan formation. If no plan or an incomplete plan is produced - i.e. at least one unit agent has no goal set - additional goals are collected from the agents and a new plan formation round is initiated.

3.2.2 'WorldModel' - The world model

The WorldModel uses the jDip world model, where provinces are divided into one or more Locations, one for every coast. The world model contains of all position data, together with distance matrixes for fleets and armies. The world model is initialized as soon as the bot connects to the server and is told what map is to be played. In the initial drafts, the world model contained a static evaluation of every province based on map location, but this was later removed in favor of an evaluation model where all evaluation was done by the unit agents.

3.2.3 'UnitAgent' - The unit agents

The unit agents represent the own units currently in play. At the start of the game, one unit agent is created for every starting unit by the bot engine. Further unit agents are created when the bot can build additional units, by letting a unit agent create itself at the best available location and the best available type for a new unit. The unit agents have the ability to evaluate their surroundings ('value(Province)') and create goals using those evaluations. They also have the ability to decide on a plan together if any single agent is given a list of goals.

3.2.4 'GoalList' - The goal list

The goal list is a sorted list of goals ('Goal'), where each goal contains information about value, threats ('Threat'), and which unit agent it belongs to. Each goal also keeps track of UnitAgents that could offer support ('supporters'), as well as how much support is needed for guaranteed success ('wantedSupport') and how much support is enough for it to be worth a try

('requiredSupport'). Supporters are sorted in order of the value they have evaluated their own position to. When a unit agent is required to add goals, it will add any (if any) goals previously added, as well as one new goal. As soon as an agent commits to a goal, all its goals will be removed from the list, and any agents supporting the goal will also remove their goals from the list.

3.2.5 Plan formation

The model is loosely based on the Contract Net [Smi77], with modifications to suit the domain. In the terminology of Smith, the task announcements used are the Goals created by the UnitAgents, and the bids are the support offers that are attached to those goals. In this way, every agent can (and will) act as both a Contract Net *manager* and *contractor*. Broadcasting of bids and task announcements was implemented as a version of token ring to lower the amount of communication necessary. When plan formation is initiated, the HaAIEngine queries all UnitAgents for Goals (bid requests) which are put into a sorted list in order of value (Fig. 3.8, 3.2). Then, the UnitAgents are requested to add all support (bids) they can give to the Goals in the list 3.3 - for each goal the supports are also sorted in order of cost[1]. Then, the owner of the first (best) Goal examines it to see if there has been enough support offered (as specified by requiredSupport). If enough support is not available, the goal will be removed from the list (Fig. 3.10). If there is enough support, the supporters that have offered the best support will be notified to commit to this goal, and they as well as the owning UnitAgent will purge all of their Goals from the list[2](Fig. 3.11). If all Goals are removed this way and there still are UnitAgents that has not committed to a Goal, a new round will take place(Fig. 3.5). All UnitAgents not committed to a Goal is then required to add any Goals previously added as well as one goal. The Goals that are re-added, will have a random chance of lowering their requiredSupport, based on the difference in value between the value of the Goal and the value of the last added Goal by the same UnitAgent (Fig. 3.9, 3.5).

[1] The cost of a support is the best alternative goal of the offering UnitAgent.

[2] Compare to the award message in the Contract Net - however, there are no tasks completed, and no report messages sent.

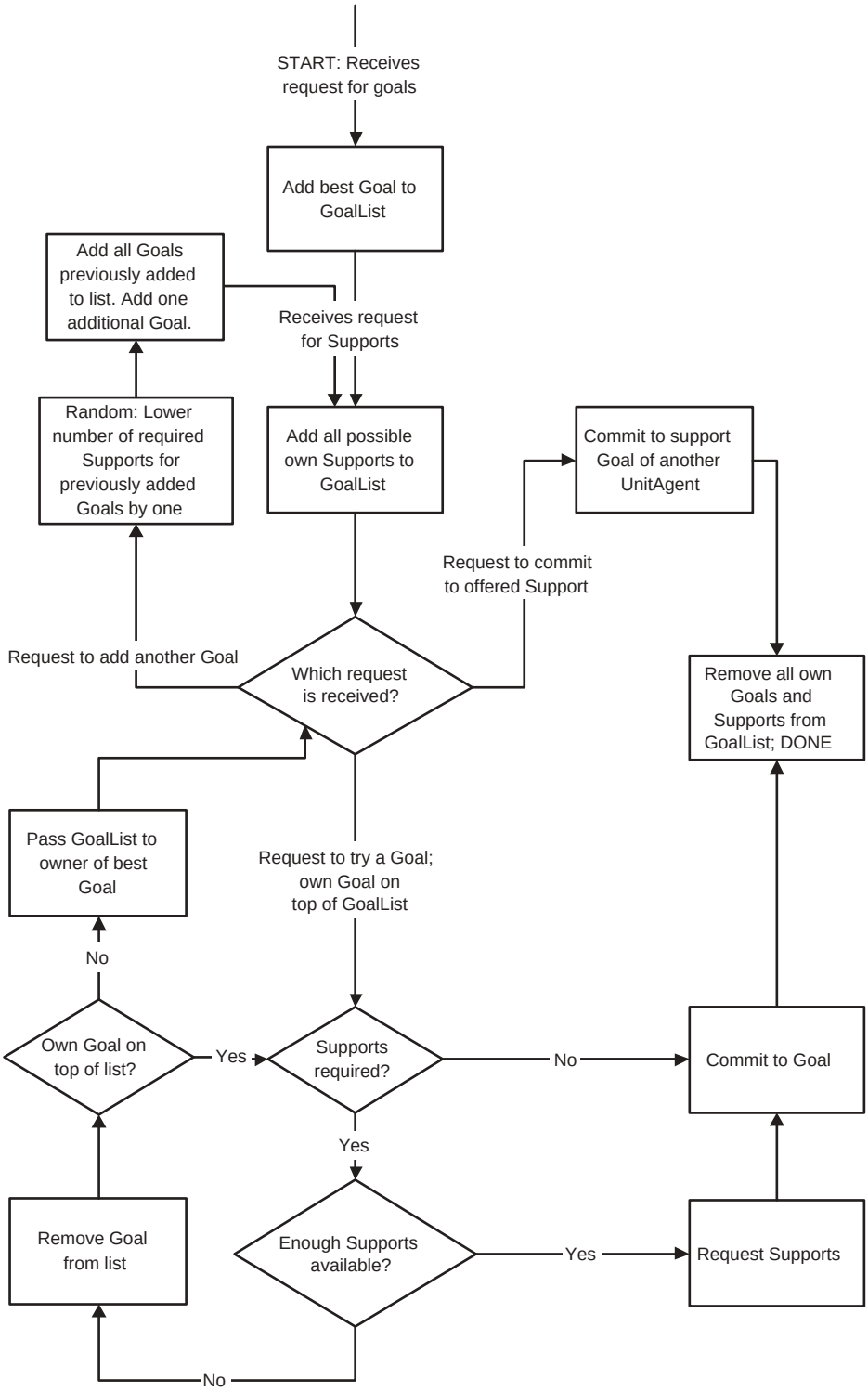

Fig. 3.1: The plan formation algorithm as seen from a single agent

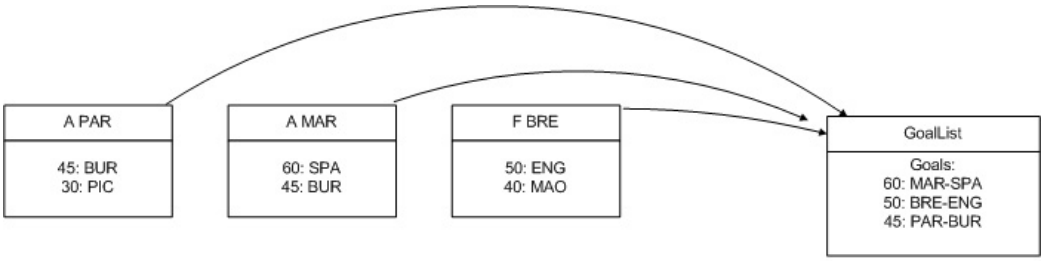

Fig. 3.2: Initial goal collection for France after evaluation is done. UnitAgents are denoted as [type] [location] - "A PAR" is an army in Paris, "F BRE" is a fleet in Brest. All UnitAgents add their best goal to the GoalList; only two goals per UnitAgent is shown here. Goals are given as value:target - the values used are for demonstration, and not reflecting the real values used by HaAI

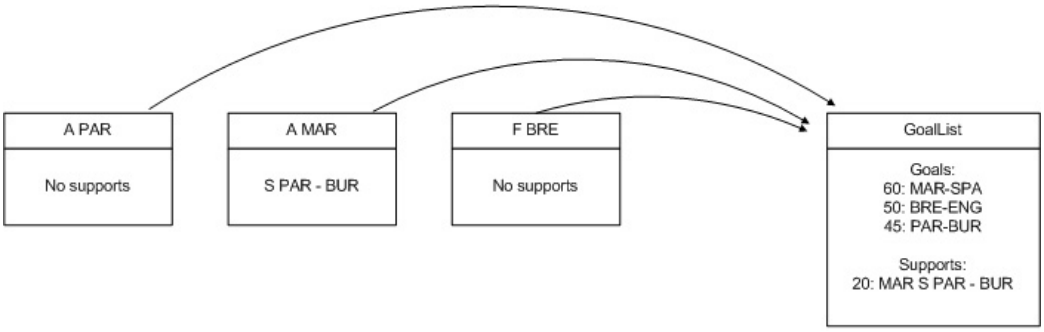

Fig. 3.3: Supports are added; only one support is possible, the army in MAR can support the army in PAR to BUR.

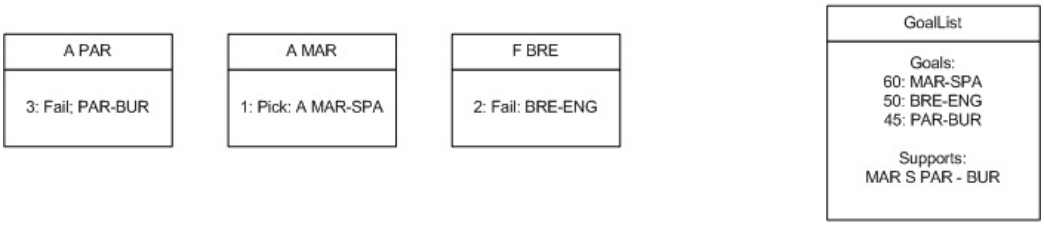

Fig. 3.4: Goals are picked: First, the army in MAR picks the move to SAP. Both F BRE and A PAR fail however, since there are threats to their moves (to ENG and BUR, respectively) and no supports are available - the support from the army in MAR was removed from the list when the goal was picked.

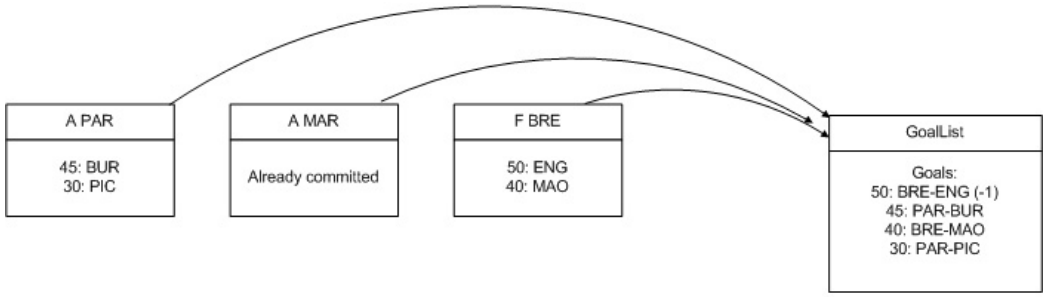

Fig. 3.5: A new round of goal collection. A MAR will not add any goals (since it is already committed). The move BRE-ENG has had its requiredSupport lowered by one when BRE-MAO was added.

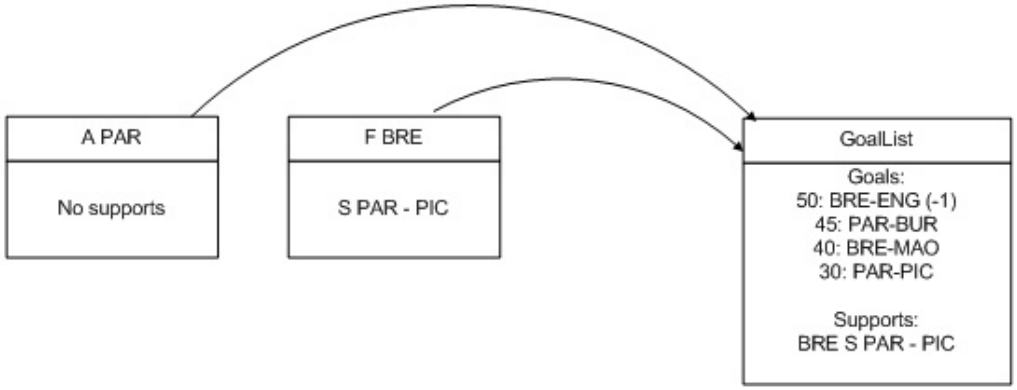

Fig. 3.6: Supports are added to the list.

Fig. 3.7: Goals are picked; The fleet in BRE picks BRE-ENG (since its required-Support has been lowered) and the army in PAR picks PAR-PIC (no support required), after trying BUR (no support available)

```
HaAIEngine:makePlan()
{
    GoalList goalList;
    do
    {
        foreach(UnitAgent agent in agentList)
        {
            agent.addGoal(goalList);
        }
        foreach(UnitAgent agent in agentList)
        {
            agent.addSupports(goalList);
        }
        goalList.first().owner().plan(goalList);

    }while(goalList.size() > 0)

    submitOrders(getOrders(agentList));
}
```

Fig. 3.8: Plan creation

```
UnitAgent:calibrateList()
{
    Goal current;
    lastValue = mLastAddedGoal.getValue();
    foreach(Goal current in addedGoals)
    {
        if (random() >
          (lastValue / current.getValue()) * (RETRY_FACTOR / 100))
        {
            current.requiredSupport--;
        }
    }
}
```

Fig. 3.9: Calibration of the GoalList to enable retries - called once every time addGoal is called.

```
UnitAgent:plan(GoalList gList)
{
    Goal goal = gList.first();
    if(goal.owner() != this)
    {
        goal.owner().plan(gList);
        return;
    }
    if(goal.requiredSupport > number of possible supporters)
    {
        gList.removeFirst();
        goal.owner().plan(gList);
        return;
    }

    //while we have not yet reached wantedSupport
    //and there are more supporters available
    while(goal.draftedSupport < goal.wantedSupport &&
            goal.supporters().size() > 0)
    {
        //Tell this UnitAgent to commit to this goal
        goal.bestSupporter().commit(goal, gList);
        this.commit(goal, gList);
    }

    gList.removeFirst();
    goal.owner().plan(gList);
}
```

Fig. 3.10: The UnitAgent planning algorithm.

```
UnitAgent:commit(Goal goal, GoalList gList)
{
    this.order = goal.getOrder(this); //Set own order
    gList.purge(this); //remove all own goals and supports from list
}
```

Fig. 3.11: Committing to a goal.

Builds

Builds are handled by the unit agent to be built. First, it will decide wether an army or fleet is most needed; it will always try to make the army/fleet ratio match the ratio of non-home centers reachable by an army/fleet. Then, it will build that kind of unit at the best available home center for that kind of unit. If attempting to build a fleet, and no fleet builds are possible, it will try to build an army instead.

Disbands

When disbanding, the unit at the position with the lowest value will always be disbanded.

Retreats

Retreats are decided by allowing any retreating units to pick their 'best' retreat, conflicts are handled based on second best retreat available.

3.2.6 Evaluation

When evaluating a province, only those provinces with a supply center have their "own" value. That value is based on whether the bot is owner of the center or not, as well as what season it is. No value is given to owned centers that are unthreatened (Table 3.1 and Figures 3.12 and 3.13). The value of provinces is also increased with a fraction (RANGE_DECLINE) of their neighbors value. In addition to this, to avoid "chains" of several moves that depend on each other, which generally is a bad thing in Diplomacy, the value for unthreatened province occupied by a friendly unit is decreased (using O_Occ_Threat and O_Occ_Unthreat). An example of such a chain is in Figure 2.3, where the army in Moscow bounces on the friendly fleet in Sevastopol, since that fleet bounces with the Turkish fleet in the Black Sea. A better move might have been Moscow-Ukraine, since that move is guaranteed to succeed. Note that a province is evaluated as separate *Location*s - each possible fleet or army position is a Location. Thus, Brest has two Locations (coast and land) and Spain has three (nc, sc, and land) while the Baltic Sea has one (sea) just as Burgundy (land).

```
UnitAgent:evaluate(Location pLoc)
{
    value = value(pLoc);

    for (int i = 1; i <= EVAL_SCOPE; i++)
    {
        foreach(Location at distance i)
        {
            if(pLoc not visited)
            {
                value += value(location) *
                    Math.pow(RANGE_DECLINE / 100, i);
            }
        }
    }

    if(pLoc has friendly unthreatened unit)
        value *= OWN_OCCUPIED_UNTHREATENED / 100;

    return value;
}
```

Fig. 3.12: The evaluation algorithm.

```
UnitAgent:value(Location pLoc)
{
    if (pLoc is not a supply center) return 0;
    value = 0;

    if(pLoc is own supply center)
    {
        if (pLoc is threatened)
        {
            if(This is a fall turn)
                value += OWN_CENTER_THREATENED_FALL;
            else
                value += OWN_CENTER_THREATENED_SPRING;
        }
        if(pLoc is home center) value += OWN_HOME_CENTER;
    }
    else
    {
        if (This is a fall turn)
            value += ENEMY_CENTER_FALL);
        else
            value += ENEMY_CENTER_SPRING;

        if(pLoc is home center) value += ENEMY_HOME_CENTER;
    }
    return value;
}
```

Fig. 3.13: The algorithm evaluating a single location.

3.2.7 Weights & Variants

HaAI was created in two different variants, with slightly different characteristics, listed in Table 3.1.

Parameter	Explanation	Van.	Ber.
EVAL_SCOPE	How far the agent should look in its evaluation.	8	12
RANGE_DECLINE	How many percent of the value of a province that should transfer to neighboring provinces.	12	10
E_C_FALL	The value of an enemy center in the fall.	1000	1200
E_C_SPRING	The value of an enemy center in the spring.	800	1000
O_C_THREAT_FALL	The value of an owned, threatened center in the fall.	1000	900
O_C_THREAT_SPRING	The value of an owned, threatened center in the spring.	800	800
O_H_CENTER	Extra value given to own threatened home centers.	1000	1200
E_H_CENTER	Extra value given to an enemy home center.	500	1000
RETRY_FACTOR	Retry tweak factor.	1.05	0.90
O_OCC_UNTHREAT	Chain avoidance factor. (Unthreated units)	0.60	0.30
O_OCC_THREAT	Chain avoidance factor. (Threatened units)	0.70	0.50

Tab. 3.1: HaAI parameters and values.

Vanilla

The basic weighting, this bot made little difference between offense and defense.

Berserk

The Berserk bot prioritize centers owned by enemies, and especially their home centers. It is more likely to try a move with low chances of success than the vanilla version.

3.3 Experiment setup

The experiments were conducted as a series of games where all seven participating bots were assigned their starting player uniformly distributed at random. Each game was played until there were

1. a solo victory,
2. no change in supply center ownership had taken place for five consecutive game years, or
3. three minutes had passed.

592 games were played on a single AMD 1700xp 768 DDR running Windows XPpro. The scores awarded were (7/(number of survivors)) points for a draw, and 7 points for a solo victory. Less than 2% of the games were aborted because of the time limit.

3.3.1 Participants

Both of the two described variants of HaAI were entered into the tournament, as well as Man'chi AttackBot 7, Man'chi DefenceBot 7, RandBot, DiploBot v1.2 and DumbBot v2.

3.4 Experiment results

In this section the results of the tournament will be presented, regarding the scores of the bots, their ability to survive an elimination and their speed.

3.4.1 Scores

There are two ways of scoring in Diplomacy, as discussed in Section 3.3: to win games (i.e. to *solo*), or to still be in the game when it ends prematurely (i.e. be part of a *draw*). In Tables 3.2–3.3 we see the individual sums of scores of the bots of the 592 matches that were run (of which 487 were solos, the rest of them ended in draws). The total score is shown in Table 3.4.

Clearly, HaAI Berserk performs well, winning 18.4 percents of the games, although DiploBot, DumbBot, HaAI Vanilla and Man'chi AttackBot all are between 12.7 and 16 percent (see Table 3.2). Although Man'chi DefenceBot was only able to win 40 matches, it was best on the draws, closely followed by HaAI Vanilla. DumbBot was by far the bot that was least able to survive to a draw, reaching barely over the level of RandBot as shown in Table 3.3.

In total, HaAI Berserk performed best with DiploBot and HaAI Vanilla on second and third place. As expected, random movements (represented by RandBot in our tournament) is not a successful strategy in this game (either).

3.4.2 Elimination

Another aspect of Diplomacy is the ability to survive. Even though a solo implies that it is just a matter of time before the soloing player will be able to conquer the rest of the provinces, it may still be of interest to see what bots are able to survive an elimination. In Table 3.5 we see that the very same bots that are among the best in reaching draws (the Man'chis and the HaAIs), also survive elimination to a higher degree than e.g. DumbBot.

3.4.3 Performance

Each bot was set to play against itself for ten minutes of effective game time, and the total number of orders submitted per second for the seven participating bots was recorded. In Table 3.6 we can see that the DumbBot was by far the fastest bot, leaving HaAI and Man'chi far behind. Slowest of them all was DiploBot, only managing 3.7 orders per second, about 15 times as slow as DumbBot.

Bot	Matches won	Percent	Solo score
HaAI 0.63 Berserk	109	18.4	763
DiploBot 2	95	16.0	665
DumbBot 2	87	14.7	609
HaAI 0.63 Vanilla	81	13.7	567
Man'chi AttackBot 7	75	12.7	525
Man'chi DefenceBot 7	40	6.8	280
RandBot	0	0	0

Tab. 3.2: Solo score totals.

Bot name	Draw Score
Man'chi DefenceBot 7	164.27
HaAI 0.63 Vanilla	157.64
Man'chi AttackBot 7	133.32
HaAI 0.63 Berserk	103.05
DiploBot 2	98.10
DumbBot 2	42.34
RandBot	36.29

Tab. 3.3: Draw score totals

Bot name	Total Score	Score per match
HaAI 0.63 Berserk	866.05	1.46
DiploBot 2	763.10	1.29
HaAI 0.63 Vanilla	724.64	1.22
Man'chi AttackBot 7	658.32	1.11
DumbBot 2	651.34	1.10
Man'chi DefenceBot 7	444.27	0.75
RandBot	36.29	0.06

Tab. 3.4: The total and average scores for the participating bots in the 592 played games.

Bot	Matches eliminated	Percent eliminated
HaAI 0.63 Vanilla	59	10.0
Man'chi DefenceBot 7	76	12.8
HaAI 0.63 Berserk	87	14.7
Man'chi AttackBot 7	115	19.4
DiploBot v1.2	116	19.6
DumbBot 2	348	58.8
Davids RandBot 1	530	89.5

Tab. 3.5: Elimination records

Bot type	Orders/s	Implementation Language
DumbBot 2	55.9	C++
HaAI 0.63	19.6	Java
Man'chi 7	12.7	Java
DiploBot 1.2	3.7	Java

Tab. 3.6: Bot performance measured in number of orders per second.

4. DISCUSSION

There is at least one possible source of errors in the results; the fact that games were aborted with no score recorded if a game dragged on for more than three minutes. While this did not happen very often, it might have rendered worse results for DiploBot, which becomes quite slow once it gets many units on the board. However, these errors are marginal since approximately 3% of the games were aborted, and most of these were server crashes at launch (in the seeding round, out of 33 games without a result logged, 32 was because of a server crash).

4.1 Parameters

The parameters used were the results of the study of other bots, the authors experience in Diplomacy, and very limited testing. It is therefore likely that these values are not optimal - in fact, it is probable that these are not even the most optimal parameters to use at all.

4.2 Model

The decision model used works well, within certain limits imposed by the design and implementation. It is efficient and coordinated [BG88] in plan formation for units at the front, and can easily be tweaked to create different characteristics. However, optimization of the algorithm and addition of new factors often required rewriting not only in the unit agents, but also in associated data types.

Another issue with the distribution of plan formation was the fact that evaluation of decisions made debugging hard because of the sheer amount of entities, data and actions involved. While this does not directly impair the bot itself, it made the development progress slower than might otherwise have been the case. This problem with traceability is to no small extent a design flaw, which may be mitigated by a design that takes this into account.

The unit agent model seems to be an efficient way of distributing the analysis and plan formation for Diplomacy, and the token ring communication limited overhead to a great extent compared to broadcast/direct communication designs also contemplated. The evaluation scheme worked reasonably well, but it is possible that a 'cleaner' solution where only the strengths of a position are included in the basic evaluation would have been more appropriate. This would require specific logics to handle logistics behind the front line.

4.3 Strategy & Tactics

Since HaAI does not plan ahead, and makes no strategic evaluations, it is all the more dependent on effective tactics. The algorithm used is developed to maximize the *immediate* utility, rather than setting up for future gains. In the domain of Diplomacy, this might not be a drawback, since the uncertainity of any plan dependent on the actions of several independant actors makes it hard to compute expected utility with any degree of precision. Still, while detailed planning might not be possible or even desireable, overall performance could probably be increased by paying attention to strategic aspects of the game.

5. CONCLUSION

The distributed approach used by HaAI was shown successful, even if it did not outperform the opposition by any large margin. Even though the bot itself is still quite simple compared to Man'Chi or Diplobot, it delivered better results and performance. While lacking the ability to plan ahead, the distributed algorithm manages to coordinate units and create a coherent plan for the immediate future. The agent approach, with rather simple agents and a straightforward protocol for the agents to coordinate their movements, keep the design of the individual parts simple and allows the complex problems of Diplomacy tactics to be divided into manageable parts.

6. FUTURE WORK

There are a number of areas where HaAI and the multi-agent approach to Diplomacy can be improved.

6.1 Strategic analysis & Threat assessment

One area where HaAI could be improved a lot, is to add strategic analysis and threat assessment. A few relatively simple ways to do this would be to analyze which centers are most likely to be part of a solo to determine targets, and use the related notion of Fear Factor [Win99][Tuf00] to spot potential threats. Another aspect of strategic analysis is the fact that powers may be more or less good targets, or important to attack. It is quite obvious that it is important to attack a power that has taken a large number of centers, and thus is close to winning the game. Furthermore, it might be of a greater value to take the last center of a power, and thus eliminate it, than it is to take the fourth or fifth center of another power.

6.2 Tactical analysis

The tactical analysis in HaAI does not try to analyze the enemy position and probable moves; instead it treats threats and enemy units in an abstract way. If one could with any certainty determine the most likely enemy targets, optimization of the tactical analysis would no doubt be possible.

6.3 Logistics

It seems that the problems facing units far behind the line is quite dissimilar to the once at the front. Handling of logistics seems to be an important part, and should probably not be left to the generic tactical analysis algorithm, but rather have an algorithm of its own to optimize performance.

6.4 Extended negotiations & Plan optimization

The agent and negotiation model needs to be extended to accommodate these changes, and ideally to be able to add new aspects, such as a negotiation module. Also, with the current implementation the optimization of plans is quite crude, especially with regards to selecting support - this could be amended either in a new negotiation model or within the current model.

6.5 Parameter optimization

Since the parameters used for the HaAI versions are the results of very limited testing, there is probably much room for improvement here. For example, genetic algorithms could be used to find good bot parameters within the framework.

BIBLIOGRAPHY

[arc04] The diplomacy archive, http://www.diplomacy-archive.com, 06 2004.

[BG88] Alan H. Bond and Les Gasser. *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann, August 1988.

[byl04] Jac page, http://www.ludd.ltu.se/ heb/dip/, 06 2004.

[Cal00] Alan B. Calhamar. *The rules of Diplomacy*. The Avalon Hill Game Company, 4th edition, 2000.

[dai04] Daide web page, http://www.starblood.org/daide/, 06 2004.

[Fra03] Henric Fransson. Agentchess - an agent chess approach. Master's thesis, Blekinge Institute of Technology, 2003.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Serach, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[jdi04] jdip web page, http://jdip.sourceforge.net, 06 2004.

[KL95] Sarit Kraus and Daniel Lehmann. Designing and building a negotiating automated agent. *Computational Intelligence*, 11(1):132–171, 1995.

[LH92] Daniel E. Loeb and Michael R. Hall. Thoughts on programming a diplomat. *Heuristic Programming in Artificial Intelligence*, 3: The Third Computer Olympiad:123–45, 1992.

[Loe95] Daniel E. Loeb. Challenges in multi-player gaming by computers. *The Diplomatic Pouch Zine*, S1995M, 1995.

[man04] Man'chi web page, http://ca.geocities.com/bmr335/manchi.html, 06 2004.

[Now96] Richard J. Nowakowski, editor. *Games of No Chance*, pages 451–471. Cambridge University Press, 1996.

[pou04] The diplomatic pouch, http://www.diplom.org, 06 2004.

[Rab02] Steve Rabin, editor. *AI Game-Programming Wisdom*. Charles River Media, 2002.

[RN95] Stuart Russel and Peter Norvig. *Artificial Intelligence*. Prentice Hall Inc, 1995.

[Sha78] Richard Sharp. *The Game of Diplomacy*. 1978.

[Smi77] R. G. Smith. The contract net: A formalism for the control of distributed problem solving. 1977.

[Tuf00] Ole R Tuft. Geographical destiny revisited. *The Diplomatic Pouch Zine*, W2000A, 2000.

[Win99] Paul D. Windsor. Geography is destiny. *The Diplomatic Pouch Zine*, F1999R, 1999.

[Woo02] Michael J. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley and Sons Ltd., 2002.