Strategy Evolution and Resolution Deduction in Diplomacy

Berto Booijink

Strategy Evolution and Resolution Deduction in Diplomacy

Berto Booijink

May 2005

Master's thesis

Artificial Intelligence
University of Groningen

RuG

Supervisor: Dr. L.C. Verbrugge
Referee: Prof. Dr. L.R.B. Schomaker

Abstract

A translation of this abstract in Dutch starts on the next page.

Diplomacy is a strategic game for seven players. Each player represents a European empire in the early years of the twentieth century with which he tries to conquer Europe. The players have the disposal of armies and fleets (units) to achieve this goal. The game proceeds in rounds. Each round all players simultaneously reveal orders for their units. All orders together determine which are actually carried out and which are not; orders may hinder or support other orders. Usually, Diplomacy players have the opportunity to negotiate with each other. This work focuses on a variant of Diplomacy, *no-press*, in which negotiation is not allowed.

Game theory is a research area in artificial intelligence that investigates the interaction between human beings. Party games provide an excellent domain for such research. Games with large search spaces are particularly interesting. Diplomacy surpasses even Go in this regard, so classic search algorithms do not stand a chance in Diplomacy. More intelligent techniques are required to fathom Diplomacy.

This work aims at logic-based Diplomacy order processing and at evolutionary Diplomacy strategy forming. To this length the goal is to develop a logic-based resolution model and an evolutionary player model with the following specifications: the resolution model must determine the correct board state, resulting from any set of orders, within insignificantly small response times, compared to those of human players. The player model must perform better than a random playing model, with response times of less than five minutes. A logic language was designed to describe orders and other aspects of the game.

This thesis covers the development of the Diplomacy resolution model 'Atlas', that processes Diplomacy orders by using logical deduction. This model passes through a number of stages in which logic compounds of growing complexity are deduced, until the solution is explicitly known. In this manner, more complex deduction techniques are only applied to more complex cases. Ares allows for the simulation of Diplomacy games and enables player models to foresee the consequences of orders.

The resolution model Atlas was tested on a set of 153 game situations that is assumed to include cases of all complexities. Atlas produced the correct resolution in all cases, given the restriction on orders to always be complete and correct. During 128 game simulations Atlas resolved 13323 game situations in an average of 8.8 milliseconds per resolution. Simpler cases were resolved faster than more complex cases.

Atlas is accurate and efficient, given the restrictions, and thereby complies with the stated specifications. Logical deduction is a profound basis for Diplomacy resolution and possibly also for logistics and management problems. Decisions would then need to be represented with multiple options, instead of binary. Future research should show the attainability of such applications.

This thesis also describes the design of an artificially intelligent Diplomacy player model, based on evolutionary computing on strategies. The model represents action alternatives for the situations it is expected to meet. The genetic fitness of action series is determined by the evaluations of the game situations that those series bring forth. The model repeatedly creates new action alternatives by mutating actions in the fittest series. In this manner, the best action series is gradually improved. A simultaneous process searches for actions of the opponents. These counter-actions are mutated to yield game situations with low evaluations; the model assumes that its opponents will continuously counter him.

The consistency of Ares' actions was investigated by repeatedly making this model generate opening actions, playing 'Germany'. A relation with the most popular openings for the same empire, as observed during internet games between human players was not found. Ares was

set up against a random playing model in 128 Diplomacy games. In each game a unique combination was used to assign the two models to the seven empires. Each game was ended when one of the players reached a victory (109 games) or after 218 rounds (19 games). In the case of a victory, the winner takes one point and in pre-ended games, one point was split equally between the survivors. Ares collected 123.0 points (96.1% of the available points) against 5.0 points (3.9%) for the random playing model. With the used parameter settings in the game simulations (a produced strategy with depth two, of the tenth generation) Ares has response times of approximately one minute.

Ares plays better than a random playing model, within five minutes per action and thereby complies with the stated specifications. Evolutionary computing is a hopeful technique in automated strategy forming. Possibly, this technique is better applicable to games in which the imperfectness of information is lower than in Diplomacy, like Stratego or Scotland Yard. The application of strategy evolution to Diplomacy leaves many possibilities for improvement. We could try to combine the intentions for similar game situations. Also, we could investigate the influence of trust and negotiations on strategy forming in standard Diplomacy, where players *are* allowed to negotiate. Finally, it might be interesting to investigate combinations of evolutionary computing with other promising AI Diplomacy playing techniques, like evaluation-based and goal-based approaches. The former tries to move units towards highly evaluated areas and the latter generates attainable goals and chooses the best possible combination of goals to pursue.

Samenvatting

Diplomacy is een strategisch spel voor zeven spelers. Elke speler vertegenwoordigt een Europees rijk aan het begin van de twintigste eeuw, waarmee hij Europa probeert te veroveren. De spelers hebben legers en vloten (eenheden) tot hun beschikking om dit doel te bereiken. Het spel verloopt in ronden. Elke ronde onthullen alle spelers gelijktijdig welke zetten (orders) ze met hun eenheden willen doen. Alle orders samen bepalen welke orders daadwerkelijk worden uitgevoerd en welke niet; orders kunnen elkaar verhinderen of juist ondersteunen. Doorgaans zijn Diplomacy-spelers in de gelegenheid met elkaar te overleggen. In dit werk ligt de nadruk op een variant van Diplomacy, *no-press*, waarbij overleg geen rol speelt.

Speltheorie is een onderzoeksgebied binnen de kunstmatige intelligentie dat zich bezig houdt met de wisselwerking tussen mensen. Gezelschapsspellen vormen een perfect domein voor dergelijk onderzoek. Spellen met grote zoekruimten zijn met name interessant. Diplomacy overtreft zelfs Go wat dit betreft, waardoor het klassieke zoekalgoritmen geen enkele kans laat. Slimmere technieken zijn nodig om Diplomacy te doorgronden.

Dit werk richt zich op de verwerking van Diplomacy-orders op basis van logische deductie en op de vorming van Diplomacy-strategieën op basis van genetische algoritmen. Hierbij is de ontwikkeling van een logica-gebaseerd resolutiemodel en een evolutionair spelermodel met de volgende eisen als doel gesteld: het resolutie-model moet voor elke combinatie van orders de juiste resulterende bordsituatie bepalen, in een verwaarloosbaar korte tijd ten opzichte van de denktijd van menselijke spelers. Het spelermodel moet Diplomacy beter spelen dan een willekeurig spelende speler, met een responstijd van maximaal vijf minuten. Een logicataal is ontwikkeld om orders en andere aspecten van het spel te beschrijven.

Deze scriptie behandelt de ontwikkeling van het resolutiemodel 'Atlas', dat Diplomacy orders verwerkt door middel van logische deductie. Dit model volgt een aantal stadia waarin steeds complexere logische constructies worden gededuceerd, totdat de oplossing expliciet bekend is. Op deze manier worden complexere deductietechnieken alleen toegepast op complexere situaties. Ares maakt simulaties van Diplomacy-spellen mogelijk en stelt spelermodel-

len in staat gevolgen van orders te voorzien.

Het resolutiemodel Atlas is getest op een set van 153 spelsituaties, waarvan wordt aangenomen dat ze gevallen van elke complexiteit omvat. In alle gevallen gaf Atlas de juiste resolutie, uitgaande van de eis dat orders altijd volledig en correct worden ingegeven. Gedurende 128 spelsimulaties bepaalde Atlas de resolutie van 13323 situaties in gemiddeld 8.8 milliseconden per resolutie. Eenvoudigere situaties werden sneller opgelost dan complexere situaties.

Atlas is correct en efficiënt, onder de gestelde voorwaarden, en voldoet daarmee aan de gestelde specificaties. Logische deductie is een gedegen basis voor Diplomacy resolutie en wellicht ook voor logistieke en beleidsmatige problemen. Beslissingen zouden in dat geval met meerdere keuzemogelijkheden moeten worden gerepresenteerd, in plaats van tweezijdig. Toekomstig onderzoek zou de haalbaarheid van dergelijke toepassingen uit moeten wijzen.

Deze scriptie beschrijft tevens het ontwerp van een kunstmatig intelligent model van een Diplomacy-speler, gebaseerd op een genetisch algoritme dat strategieën evolueert. Het model representeert actie-alternatieven voor de situaties die hij verwacht tegen te komen. De genetische fitheid van series van acties wordt bepaald door de evaluatie van de spelsituatie die uit die series voortvloeit. Het model creëert herhaaldelijk nieuwe actie-alternatieven door mutatie van de acties in de fitste serie. Op deze manier wordt de beste serie van acties geleidelijk verbeterd. Een gelijktijdig proces zoekt naar tegenwerkende acties van tegenstanders. Deze tegenacties worden gemuteerd zodat ze spelsituaties met lage evaluaties opleveren; het model neemt aan dat zijn tegenstanders hem voortdurend zullen proberen tegen te werken.

De consistentie van Ares' acties is onderzocht door dit model herhaaldelijk openingsacties te laten genereren, spelend met 'Duitsland'. Een relatie met de meest populaire openingen voor hetzelfde rijk, geobserveerd gedurende internetspellen tussen menselijke spelers werd niet gevonden. Verder is Ares in 128 Diplomacy spellen opgezet tegen een model van een willekeurig spelende speler. In elk spel werd gebruik gemaakt van een unieke combinatie van toewijzingen van de twee modellen aan de zeven rijken. Elk spel werd beëindigd zodra één van de spelers een overwinning had behaald (109 spellen) of na 218 ronden (19 spellen). Een overwinning leverde een punt op en bij een vroegtijdige beëindiging werd één punt gelijk verdeeld onder de overlevenden. Ares behaalde een totale score van 123.0 (96.1% van de te behalen punten) tegen 5.0 punten (3.9%) voor het willekeurig spelende model. Bij de gebruikte parameterinstellingen voor de spelsimulaties (een opgeleverde strategie tot diepte twee, van de tiende generatie) heeft Ares een responstijd van ongeveer één minuut.

Ares speelt beter dan een willekeurige model, binnen vijf minuten per actie, en voldoet daarmee aan de gestelde specificaties. Genetische algoritmen zijn een hoopvolle techniek in automatische strategievorming. Mogelijk is deze techniek beter toepasbaar op spellen waarbij de informatieonzekerheid lager is, zoals Stratego of Scotland Yard. In de toepassing van strategie-evolutie op Diplomacy zijn nog vele verbeteringen mogelijk. Men zou kunnen proberen om intenties voor vergelijkbare spelsituaties met elkaar te combineren. Voorts zou men de invloed van vertrouwen en onderhandelingen op strategievorming kunnen onderzoeken in standaard Diplomacy, waarbij spelers *wel* mogen overleggen. Tot slot is het wellicht interessant combinaties te onderzoeken van genetische algoritmen met andere veelbelovende technieken in AI Diplomacy, zoals evaluatie-gebaseerde en doel-gebaseerde benaderingen. De eerste tracht eenheden naar hooggewaardeerde gebieden te verplaatsen en de tweede genereert haalbare doelen en kiest de beste combinatie van doelen om na te streven.

Acknowledgements

Thank you for educational grounds, moral support, language guidance, technical assistance, much patience, the tosti special, and many games of ping-pong:

Rineke Verbrugge, Lambert Schomaker, Johanneke Siljee, Hetty and Herman Booijink, Carien Booijink, Douwe Terluin, Marleen Schippers, Jan-Willem Marck, Gerben Blom, Liesbeth van der Feen, Hein van der Ploeg, Tom ten Thij, Mathijs de Boer, Daan Reid, Jan-Willem Hiddink, Robert Coehoorn, Ronald Zwaagstra, Pim Dorrestijn, Lisette Mol, Pim van Oerle, Nina Schaap, Maaike Harbers, Hendrik Jan de Jong, Martijn Pronk, Manda Ophuis, Chris Postma, Sonny Onderwater, Douwe Egberts, Harald and Erik Mimpen at Mihosnet

Contents

1 Introduction

Throughout the years, game playing has presented artificial intelligence with many challenging problems. In general, the aim has been to create an agent that can compete with human players in a particular game. For some games a strategy has been found and proven competitive. In Chess, for instance, artificial intelligence became world champion. A search algorithm investigates possible continuations of the game and concludes sensible moves. Applying current search techniques to games with a smaller search space, like Tic Tac Toe or Nim, has lead to complete strategies that guarantee the best move in each game situation [4].

The way that artificial intelligence solves game problems need not necessarily be similar to human approaches to those games. Both human and artificial players do what they do best and are likely to win games that fit their way of thinking. In games with very few possible game states, like Tic Tac Toe, an artificial agent might remember what to do in every possible case. It will not needlessly lose since its algorithm suits the problem so well. Humans would still be likely to apply reasoning to improve their chances, which is less promising in this case.

Today many aspects of game playing remain unmatched. Games that still frighten an artificial agent have one thing in common: the search space is huge. Examples are Go, Dots and Boxes, and Diplomacy. There is no way that any computer in the near future could compare all possible moves in a game situation and present an appropriate move fast enough to fit our patience. Those games require a much more efficient approach to find solutions. In their quest for artificial intelligence, researchers often crib from the pool of best examples available: nature. For instance, in Go it has been proposed that humans try to recognize patterns when they play the game. The extension of this approach to computer algorithms has produced promising results. This thesis will stretch it just a little bit more.

1.1 Problems in artificially intelligent Diplomacy playing

Diplomacy is a game that holds some unconventional properties that impose the need for interesting new roads in the game playing world. A few of these properties are described here, to help the reader understand why Diplomacy is in fact a problem child.

Whereas in most games players take turns, Diplomacy involves the simultaneous moves of all players. This makes the application of traditional search trees impossible, since there is no way of choosing who should go first in the search tree.

Even if we would find a way to represent moves of different agents in one search tree, the tree would be incredibly large. The game of Go has 361 opening possibilities, resulting in about 10^{20} nodes when looking four own moves ahead. Diplomacy has 4.430.690.040.914.420 possible openings (for all seven players) [18] and an estimate of at least 10^{62} nodes to plan four moves. Knowing we cannot search a tree for Go, we should not try to do so for Diplomacy either.

Most games are competitive (Chess, Poker, Tic Tac Toe e.a.), meaning that agents need to counter others to gain themselves. Some games are co-operative (Lord of the Rings), where agents need to join forces to win together. Diplomacy is a game of co-opetition, meaning that agents have conflicting goals, but need to co-operate to reach those goals [7]. This property inflicts another layer of strategy upon the game, involving negotiations, trust, and diplomacy.

Not only strategy forming is hard to formalize. The manner in which actions are processed deserves some attention too. The success of an action in chess solely depends on how the particular piece may move and the state of the field to which it tries to move. An agent can even predict the outcome of his actions and always make successful (allowed) actions. In Diplomacy, agents mostly do not know what their actions bring about since the success of an action also depends on other actions at that time. In fact, the success of all agents' actions depends on themselves other according to a complex web of rules, making their adjudication all but trivial.

1.2 The game of Diplomacy

Diplomacy is a game of negotiations, alliances, promises kept and promises broken. To survive, a player needs help from others and to win the game, a player must eventually stand alone. Knowing whom to trust, when to trust them, what to promise and when to promise is the heart of the game. The physical Diplomacy board game by Avalon Hill includes a booklet with the game rules [8]. A brief sketch is given below.

1.2.1 Diplomacy in a nut shell

Traditionally, Diplomacy is played by seven players on a game board (at a table, if you will). Each player represents one of the leading European empires in the early years of the twentieth century. The board is divided into 82 areas, some land and some sea. At the start of the game most of the land areas belong to a certain empire, as shown in figure 1.

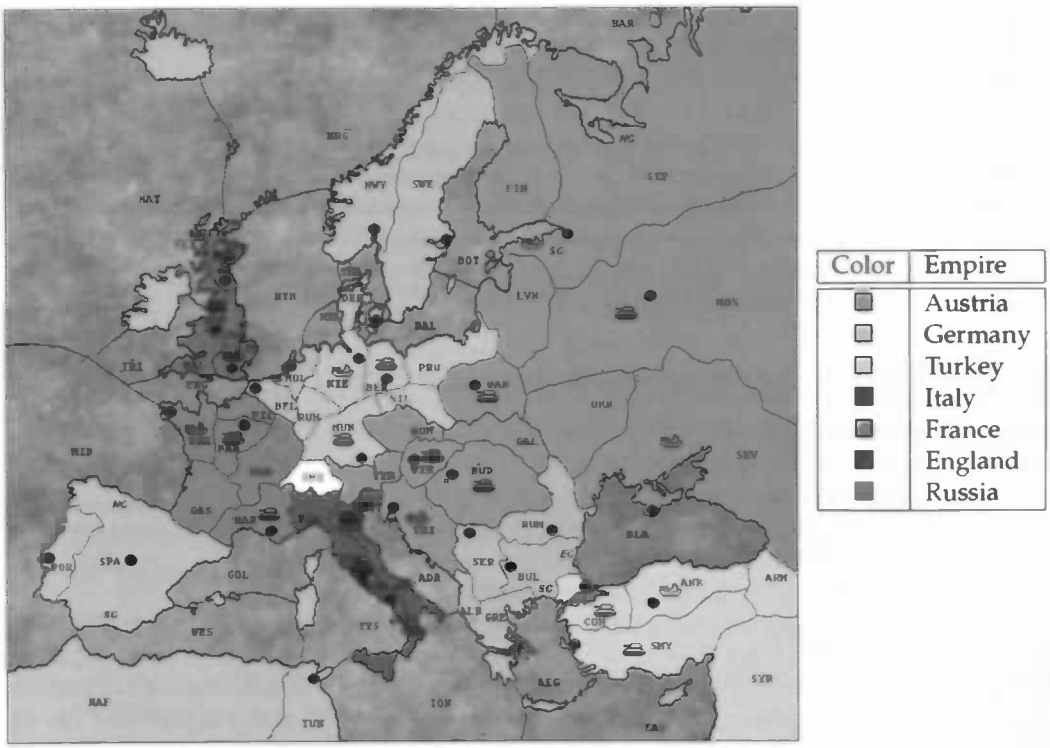

Figure 1: The initial game board

Every player must try to extend his empire using his armies and fleets. Some land areas contain production centers (cities that provide food and weapons to maintain one army or fleet each). The empire named 'Austria and Hungary' elsewhere is denoted by 'Austria' in this work.

The game proceeds in fictional years as depicted in figure 2.

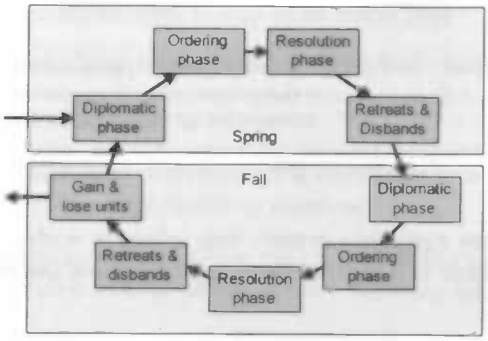

Figure 2: One year consists of two seasons with four and five phases respectively

Each year consists of two seasons, *Spring* and *Fall*. At the beginning of each season players have the opportunity to negotiate with each other on their strategy. Next, each player secretly issues an order for each of his units. One may for instance order a unit to stay in (*hold*) the area it is in or attack another area (*move*). Alternative to a *hold* or a *move*, a unit may *support* another unit's *move* or *hold*, increasing that unit's strength. In case of conflicting orders only the strongest (most supported) orders are followed (obeyed).

A fleet in a sea may be issued to *convoy* an army's *move*, enabling that army to *move* via the area of the convoying unit. A path of convoying fleets can *move* an army over the entire path. Note that a convoy *enables* a unit to move over sea, it cannot force it.

Usually, players are expected to explicitly denote a particular *move*'s use of *convoy*, distinguishing it from a possible alternative route over land. We do not demand such notation since such ambiguity seldom occurs and even more rarely results in unintended results.

At the end of each season all orders are revealed simultaneously and resolved in the following manner: one of the players calls the orders he issued, one by one. Each order is immediately followed, unless another player believes that one or more of *his* orders conflicts with it. In that case a solution to all dependent orders is sought. If a particular situation grows too complex, the players might decide to adjudicate other orders first. When a player has called all of his orders, the next player proceeds until all players have called their orders.

After resolution, players get a chance to retreat their dislodged units (if any). That is, if a unit did not move and was successfully attacked, its owner may move it to a safe, adjacent area. An area is safe if no unit tried to move to that area. At the end of each year (two seasons) each empire is extended with the areas in which units of that empire are located. The extension of one empire may cause another empire to shrink. The number of production centers each empire then controls (lie within each empire) defines the number of units that empire may own. If appropriate, players get a chance to build or disband units. A player may build units only in production center areas he originally controlled and still controls. To win, one needs to dominate Europe. Only when one empire controls more than half of the production centers, its victory is declared.

Rather often a player needs the help of other players for his orders to succeed. Such cooperation is established by means of binary or multi-sided negotiations at the start of each season. Players may reach an agreement over anything at all and may decide to break any promise at any time. Players may have private meetings excluding other players from knowledge on agreements. A good relationship with other players may greatly enhance one's chances, but to gain personally one ultimately needs to disappoint others, break promises and violate trust. Each player thus constantly needs to weigh strategic aspects against social aspects.

1.2.2 Terminology

Throughout this thesis, many technical, mostly game-specific concepts are used. For a good understanding of the text a firm grasp of these concepts is recommended.

- *Adjudication*: The process to determine which orders in a given set are followed and which are not.
- *Agent*: In general, an agent is a system that interacts with its world. Within the scope of this thesis, an agent is a human being or a machine (model) that plays the game of Diplomacy.
- *Area*: One of the 82 subsections of the board.
- *Army*: A piece (pawn), which an agent may move over land areas or via fleets to land areas oversees.
- *Board*: A representation of Europe, divided into 82 areas, on which Diplomacy is played. A board may also refer to a particular configuration of units and area ownership at a particular moment.
- *Build*: The addition of a unit to the board.
- *Coast*: A land area next to a sea area. Some land areas meet with sea areas on two sides. Those coasts are denoted by additional code names.
- *Convoy*: The order to allow an army move via a fleet's area.
- *Cut*: An unsuccessful support.
- *Defeat*: A move order defeats another order if it has more strength. In the case of multiple moves to the same area, a move should have more strength than any other move's resistance, to defeat it.
- *Disband*: The removal of a unit from the board. This happens when the unit has been dislodged and there is no area to retreat to or when the owning empire controls insufficient supply centers.
- *Dislodge*: The failure of a unit to stay in or return to its home area.
- *Disrupt*: When a convoying fleet is dislodged, the convoy path it contributed to is disrupted.
- *Empire*: Each unit belongs to one of the empires Austria, Germany, Turkey, Italy, France, England, and Russia. Each empire (and thus its set of units) is controlled by an individual agent.
- *Fail*: An unsuccessful move.
- *Fleet*: A piece (pawn), which an agent may move over seas or coasts.
- *Follow*: An order is followed if it has been legally issued and no other orders or board restrictions forbid it.
- *Foreign*: Belonging to different empires.
- *Friendly*: Belonging to the same empire.

- *Hold*: The order for a particular unit to stay in its home area.
- *Home area*: The area a particular unit is in.
- *Issue*: The commitment of an order to an empire.
- *Judge*: A judge (or adjudicator) determines the obedience to a set of orders and deduces the resulting board. He who adjudicates or resolves.
- *Legal*: The issue of an order is legal when the ordered unit exists, belongs to the issuing empire and did not receive additional issues from the same empire.
- *Move*: The order for a particular unit to move to a particular area.
- *Order*: A unit's action. An order can be a hold, move, support or convoy. In this thesis *order* is never used meaning *sequence*.
- *Production center*: A city that provides maintenance for one unit. The number of production centers that lie within a particular empire define the number of units permitted to that empire.
- *Resistance*: Each order has a resistance of one, increased by one for each support of that order.
- *Resolution*: The process to determine the result of a given set of orders on a given board state.
- *Retreat*: The obliged move of a unit to an adjacent, safe area (if any) after it has been dislodged.
- *Standoff*: Two or more moves to the same area with equal strength.
- *Strength*: Each order has strength of one, increased by one for each support of that order, excluding supports of the dislodgement of friendly units.
- *Success*: A followed order is successful.
- *Support*: The order for a particular unit to support another unit's hold or move. A valid support describes the exact order of the unit it supports.
- *Target area*: The area a particular unit attempts to move to.
- *Unit*: An *army* or a *fleet*.
- *Valid*: The applicability of a support order. A support order is valid if the supported order is legally issued.

In Diplomacy the difference between *adjudication* and *resolution* is trivial. Strictly, *resolution* refers to the consequence of a given set of orders on a given board whereas *adjudication* means the decision making on each individual order's success. In Diplomacy, adjudication thus inflicts resolution and resolution needs adjudication.

An automated order handling system is often said to perform *adjudication* since it focuses on the success of orders, whereas game descriptions speak of *resolution* since we "just want to know the next board state". Since *adjudication* and *resolution* ultimately mean the same thing (some adjustments to some board) one should not worry too much about the difference between them.

1.2.3 Resolution rules

Diplomacy demands the following rules on a correct resolution [8] (some rules were slightly changed to reduce Avalon Hill's obscurity and to meet this work's definitions):

1. *Without support all orders have the same strength.*
2. *There can only be one unit in an area at a time.*
3. *Equally supported moves to the same area cause all involved units to remain in their original area.*
4. *A standoff does not dislodge a unit already in the area where the standoff took place.*
5. *One unit not moving can stop a series of other units from moving.*
6. *Units cannot trade places without the use of a convoy.*
7. *Three or more units can rotate areas during a turn provided none directly trade places.*
8. *An order other than a move can be supported by a support order that only mentions the current area of the unit that is involved in the supported order.*
9. *A move can only be supported by a support order that matches that move.*
10. *The order of a dislodged unit can still cause a standoff in an area different from the one that the dislodging unit came from.*
11. *The order of a dislodged unit, even with support, has no effect on the area that the dislodging unit came from.*
12. *An empire cannot dislodge or support the dislodgement of one of its own units, even if that dislodgement is unexpected.*
13. *Support is cut if the unit giving support is attacked from any area except the one where support is being given.*
14. *Support is cut if the supporting unit is dislodged.*
15. *A unit being dislodged by a unit in one area can still cut support in another.*
16. *An attack by an empire on one of its own units does not cut support.*
17. *A dislodgement of a fleet necessary to a convoy causes that convoy to fail.*
18. *A convoy that causes the convoyed army to standoff at its destination results in that army remaining in its original area.*
19. *Two units can exchange places if either or both are convoyed. (This is the exception to rule 6.)*
20. *An army convoyed using alternate convoy orders reaches its destination as long as at least one convoy route remains open.*
21. *A convoyed army does not cut the support of a unit supporting an attack against one of the fleets necessary for the army to convoy. (This supersedes rule 13.)*
22. *An army with at least one successful convoy route will cut the support given by a unit in the destination area that is supporting an attack on a fleet in an alternate route in that convoy. (This supersedes rule 21.)*

1.2.4 Rule adjustments and extensions

As will be explained later, the above rules cannot adjudicate every set of orders. It also introduces synonymous notations for some orders. As a result, the following modifications were made:

Rule 8 is omitted since it introduces an alternative notation to the same order, which in our opinion is more confusing than is it convenient.

Some rules (like Kruijswijk's rules [17]) suggest to turn a blind eye to orders with unspecified coast when only one coast is possible. It is suggested that an attempt is made to follow the order as if it was issued with the correct coast specification. We are not that flexible since it only causes uncertainty with human players and any agent can kindly be asked to complete his orders.

Kruijswijk also suggests that a move via convoy should explicitly be issued to move 'via convoy' if that same move is possible over land [17]. Now the debate arises over when exactly the move order should fail. Consider an army move from Belgium to Holland. Although a clear cut route over land exists, the army decides to attack Holland through the use of a fleet in the North Sea. It should be no surprise that the move succeeds if the convoy succeeds. It also seems to be agreed upon that if there is no fleet in the North Sea, the alternative route over land is taken and the move still succeeds(!) (e.g. test case 6.G.8 [17]). However, if the convoying fleet is attacked, the convoy fails and Kruijswijk suggests that the army does not move, not even over land. We believe that this is a very farfetched distinction and it seems unreasonable to always demand the 'via convoy' specification to make it. Besides, the described situation virtually never occurs and when it does, it is very likely that the Belgian army too will take the high road.

The game rules state that if a resolution to all orders exists, that resolution is carried out [8]. However, the game rules have been proven inconsistent in some cases [25]. That is, situations exist in which the game rules result in a paradox. In such cases there might be multiple possible resolutions or no resolution at all.

The most commonly used rule to solve the very rare case of a paradox was proposed by Simon Szykman [25] as an extension to the resolution rules described by Avalon Hill [8]:

- *If a situation arises in which an army's convoy order results in a paradoxical adjudication, the turn is adjudicated as if the convoying army had been ordered to hold.*

Szykman's rule favors the success of a support over the success of a move in paradoxical convoy situations. To us it seems natural to assume a similar resolution in *all* paradoxical situations. Thus, a support should always have priority over a move when only one of them can succeed, as in the following example. Consider a real army, attempting an attack on some target. Imagine that the army cannot reach the target because of some side attack. The only way for the army to stop the side attack is to reach its target itself. The army should thus assume that the side attack is not even there, reach its target (which would stop the side attack) and then conclude that the assumption was right. This is not how real combat works.

Inspired by actual combat, we propose that the resolution rules are not extended by Szykman's rule, but by our *supports sustain* rule:

23. *Support is never cut nor dislodged by a move whose attainability depends on the success of that support (This supersedes rule 13).*

As we will see later this rule accounts for the solution to most paradoxical situations. However, we are not there yet. Suppose we have two supports that both comply to rule 23. According to rule 23 we should decide both supports to succeed. If these two decisions have

conflicting results on the outcome of one or more other orders we cannot make them without violating logic. The alert reader might notice that Szykman's rule does eliminate this kind of paradox, but not as it should. The change of the order of one paradoxical convoying army to a *hold* might enable another army to make its paradoxical *move* by convoy anyway. The outcome of resolution thus depends on where you start applying the rule. That is unacceptable.

Rule 23 states that *support* in these cases is *never* cut nor dislodged. In the described cases with two supports whose success would indirectly cause the other to fail, those supports should and will succeed, regardless of any other rule. We thus force success on the support orders at hand, inflicting an inconsistency upon one or more other orders. Both success and failure of those remaining orders would violate at least one rule.

Szykman was right about the fact that sooner or later we do need a rule that only applies in paradoxical situations [25]. Rule 23 has merely narrowed the set of paradoxical cases to the ones in which rule 23 contradicts itself. It has been suggested that when no other rule solves the resolution, no *move* in a paradoxical convoy situation should succeed, as described in the *all hold* rule:

- *If a situation arises in which an army's convoy order results in a paradoxical adjudication, all the moves part of the paradoxical situation fail* ([17] *e.a.*).

At this point (for the remaining paradoxical cases), we should take this rule. Again, the more general version is chosen over the one specialized to convoys:

24. *Paradoxical moves fail.*

Consequently, since the success of *holds* and *convoys* can only depend on the success of *moves*, paradoxical holds and convoys succeed. The proposed rules (1 through 7 and 9 through 24) thus guarantee an adjudication of all orders, always.

1.2.5 Diplomacy on-line

These days many people play Diplomacy on the internet. Usually a server is used to host the game, managing the orders and the game state. Each season the players may negotiate by whatever means necessary and at a previously agreed upon deadline all orders are resolved. In some cases the server performs the resolutions, in other cases human game masters are assigned to serve this purpose. After each season each player may retreat his dislodged units (if any) and after each year each player may build or disband units to match the number of supply centers he occupies.

Usually an automated adjudicator is used to make the resolutions. An example is the jDip adjudicator, which is said to be very fast, very accurate, and supports the latest (2000) rules [15]. Most artificial agents make no use of the adjudicator to plan their moves (as we will see in section 1.5). Adjudication times are then negligible, compared to the time agents need to write orders. However, when an agent does rely on resolution predictions, adjudicator efficiency becomes one of the most important factors in the agent's planning times.

Many variations exist on internet Diplomacy games. Some involve the restriction on negotiations, often excluding certain kinds of messages. The variant with no negotiation at all is called *no-press*.

1.2.6 Scores

In Diplomacy, although the number of production centers one owns gives a good estimate of the strength of one's empire, it should not be misconstrued with one's chances on victory. Even

when a player owns 49 percent of the production centers, all other players could join forces and still have a better chance on winning the game. Only when a player owns more than half of the production centers he is considered the winner.

Usually the property of each player's score during the game is not so important, since we only care about victory. The first player to own 18 production centers wins (gets one victory point). However, sometimes it takes too long for the game to end. Even more, the players may decide to play a fixed number of rounds beforehand. And let us not forget artificial agents who are expected to play many games, but do not get enough time to finish them all. It would be a shame if all pre-ended games were useless for ranking purposes. The common manner to determine each player's score in pre-ended games is to split the victory point equally among the survivors (e.g. [11]).

1.3 Research goals

The goal of this work is to try to design an automated resolution model (judge) and an artificially intelligent agent model. Specifications of the two models are given below. Together they should allow for many automatically played games.

1.3.1 Judge model specification and goals

The adjudication of orders should be made from the units' point of view. Each unit would draw up the restrictions that keep it from following its order. The judge should at all times bare those restrictions in mind and search for the solution that satisfies them all.

The application span of the automated resolution model should be as broad as possible. The model should be able to cope with any game situation and order set that might occur and provide the correct resolution. This has two main reasons. First, the resolution should not favor any empire over any other. It is agreed upon that the proposed rules do not, but any major derivation just might. Secondly, the quality of an artificial agent might depend on the correctness of the judge it uses. A particularly complex set of orders could be illegal and thus unthinkable for a human being, yet findable and thus playable for a computer program. An artificial agent could systematically exploit this flaw and have an advantage in playing the game. Of course a flaw in the adjudication model would also be highly unsettling for its creator.

The quality of the above described judge will be measured by facing it with a large test set of game situations and orders for which the correct resolution is known. The set contains situations with the most complex order combinations the Diplomacy community could think of. Most of them were actually constructed to tackle automated resolution systems. If the judge never fails in providing the correct resolution on the test set the research goal on its quality is reached.

The judge should be very efficient, enabling an artificial agent to foresee the outcome of many options fast. The easiest way to measure its efficiency seems to compare its resolution times with those of existing resolution systems. Unfortunately, when this work came about, there were hardly any automated adjudication systems available, let alone resolution times we could compare with. However, the average resolution time of this model might be a good estimate of its efficiency.

The efficiency of the judge will be measured by the average resolution times on boards in actual game simulations. The judge would be sufficiently efficient if its resolution times are an insignificant factor in the progress of the game and in an agent's action consideration. The judge efficiency goal would then be reached.

1.3.2 Agent model specification and goals

The agent model forms its strategies by means of evolutionary computing.

This work is mainly about the strategic aspect of the game. Since negotiations are beyond that scope, no-press Diplomacy will suffice. This variant makes the game much faster to play and thus easier to cover.

We will focus on the ordering at diplomatic phases only. Choices concerning retreats, disbands and builds are excluded from this research. However, since the game cannot proceed without them, they will be generated automatically and randomly for all agents. Since it is assumed that the strategy of the game mainly lies in the order decision making, this should not cause for major problems.

Convoy orders will not be among the options of the agent, since they hardly occur, but require for relatively complex planning. Without convoy orders an agent should be able to form strategies faster. When a proper agent model has been built, the imposed limitations could become extensions to the research domain. Note that the specification of the judge *does* include the ability to handle convoy orders.

The results of the agent model would be meaningful if they are reproducible. That is, if there is a best move, the model should choose it significantly often (more often than any other order). To validate the sensibility of results of the model, we will compare them to the frequencies of according orders in games on the internet, between human players. If many people made the same moves as the model does, the people and the model are more likely to be right. Furthermore, we will set our model up against randomly playing models and against instances of itself. Although it seems very hard to pin-point the absolute level of performance, the comparison to a random player should be a good start.

The agent model is good enough if it outperforms a random player. The agent quality goal is reached if the average score of the agent in various set-ups of Diplomacy games is significantly higher than the average score of the random player in those games.

The efficiency of the agent model should be at least competitive with human players, who are obliged to write down their orders in 5 minutes. For simulation purposes, one would wish for a much faster algorithm, but the goal of this research is set to just meet entertainment needs. That is, the goal on the agent model's efficiency is to make it act within 5 minutes time.

1.4 Automated adjudicator types

In internet Diplomacy, an automated adjudicator (judge) is mostly welcomed; in research in artificially intelligent playing it is almost indispensable. Agents do not know for sure what actions will be successful, so a judge should determine the consequences. Even more, an artificial agent model might depend on the judge in trying to foresee the consequences of its actions. Writing a Diplomacy adjudicator program may not seem to be more difficult than writing a program that checks the moves of a chess game. However, the contrary is true.

Kruijswijk describes two properties of the adjudication process that should be considered when writing the adjudication program [17]. The first property is that a set of orders leads to a set of decisions to be made (on the success of each order). One order may lead to multiple decisions. For instance, when a unit is ordered to move, and the move is decided to fail (because it was insufficiently supported), another decision emerges. To determine the influence of the unit on the area where it was ordered to move the success of the convoying unit needs to be decided. That decision would be pointless otherwise.

The second property is that the decisions depend on each other. Certain decisions can only be made when other decisions are made first. For instance, when the units are ordered to follow each other in a move, then the decision whether the unit at the end moves depends on the decision whether the unit at the front moves.

Now, to make the decisions, Kruijswijk suggests two fundamentally different methods to deal with its dependencies: a sequence-based algorithm and a decision-based algorithm. In this work two alternative methods are proposed: a condition-based algorithm and a logic-based algorithm. All four methods are discussed briefly in the next subsections.

1.4.1 Sequence-based

In a sequence-based algorithm, the program tackles the problem of dependencies in a fixed sequence. At each step of the sequence, the decisions involved should have no undecided dependencies. So, in the sequence, any decision making is preceded by the decision makings it depends on. The algorithm needs to perform all steps to ensure a correct resolution to any board. An example of a sequence-based algorithm is implemented by Black [5].

The main disadvantage of a sequence-based algorithm is that it needs to check for complex combinations of orders in every situation it is presented with to ensure that common resolution rules apply. In [5] we see that of the 20 steps of the procedure, we find "fight ordinary battles" at step 19, which does not imply high efficiency on the average cases.

1.4.2 Decision-based

In a decision-based algorithm, the program does not follow a predefined sequence of properties to tackle. The program visits the decisions (also, in no particular sequence) and tries to make them. If the decision is made, it is final. If not, the program proceeds to the next decision. This process is repeated until all decisions have been made or no more decisions can be made. An example of a decision-based algorithm is included in [17].

Compared to the sequence-based algorithm, the checks to be made when visiting a decision are very simple. The program should only check if all dependant decisions have been made and draw its conclusion, whereas the search for circular dependencies or convoy paths can be a cumbersome task. Also, the algorithm allows for decision making on the base of incomplete information when current knowledge is exhausting adjudication. That is, even when not all premises of a decision are known, the logical combination might result in one possible adjudication. For instance, when one predicate in a conjunction is known to be *false*, the conjunction is *false* regardless of the truth value of the second predicate.

A disadvantage of the decision-based algorithm is that, as with the sequence-based algorithm, each order is visited many times before it is adjudicated, probably lowering the model's efficiency. Also, it does not have a direct manner to solve paradoxical situations, or even circular dependencies. They are recognized by detecting that the algorithm has stopped. At that point it is yet to be determined what caused the stop and how to proceed.

1.4.3 Condition-based

A condition-based algorithm tries to determine the outcome of a condition as soon as a decision appears to depend on it (much like the programming language *ProLog*). The algorithm starts at whatever decision and finds a condition it may depend on. Surely this too will be a decision and the algorithm immediately tries to find that decisions success conditions. The algorithm thus immediately penetrates to the depths at which the dependencies to a decision lie.

The main advantage of algorithms of this type is that it visits any decision just once. Again, an important disadvantage is that complex dependencies cannot easily be solved. Whenever the algorithm encounters a contradiction, it has no way of knowing what caused it, unless it keeps track of the base to each decision it makes. Moreover, it might encounter a circular dependency that is not even there. That is, a decision might depend on a combination of conditions of which some can be determined and some cannot. If the algorithm bites into the second

kind, it does not have an easy way of knowing whether it really has a problem. It should go back and try other branches, which undermines the advantages of the algorithm.

1.4.4 Logic-based

The logic-based algorithm copes with the entire set of orders as a whole, but with constant knowledge of the (remaining) dependencies of each decision to be made. As in the decision-based algorithm, all conditions are specified first, but now we apply deduction to approach resolution. The algorithm constantly copes with the set of orders as a whole and makes deductions as needed.

The algorithm recognizes the solution as soon as it is explicitly present in the decision conditions. It need never check for any dependency more complex than the complexity evident from the conditions at that point. Furthermore, the dependencies of a decision remain explicit and possible restrictions to make that decision gradually become apparent as the algorithm proceeds. Even in the most complex cases like the paradoxical ones, the impossible decision(s) exactly state(s) what causes the problem.

1.5 Other approaches to Diplomacy AI

Some attempts were made to create an automated Diplomacy player before the model described in this thesis was implemented, as described by Håård [11] and Huff et al [14]:

- The Israeli Diplomat (by S. Kraus, 1995 [16])

  Primarily concerned with the diplomatic aspect of the game and was reported quite successful. The Israeli Diplomat uses an agent based approach, and distributes tasks between agents that are ordered in a hierarchical fashion [16]. Source and binaries for the Israeli Diplomat seem to be lost [11].

We doubt the validity of Kraus' research methods. All games were played via e-mail and as Kraus described herself some people definitely lost interest in playing the game [16]. Since negotiation is the most important factor in a game in which it is allowed, the Israeli Diplomat stood a better chance. Sometimes human players even forgot to write orders, making even a random guess a better strategy.

- The Bordeaux Diplomat (by D.E. Loeb, 1996 [19])

  Based on an optimized best-first search algorithm. 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 [19]. Source and binaries for the Bordeaux Diplomat seem to be lost as well [11].

- The Hasbro Diplomacy Game (by Microprose, 2000 [20])

  Based on an AI which is commonly acknowledged to be extremely poor [14].

Today's most constructive project on Diplomacy AI is called *Diplomacy AI Development Environment* (DAIDE) [24]. This project aims at the development of an artificially intelligent Diplomacy player and provides several tools to let different bots compete. It was set up at the time this work came about and has become a very flourishing project.

- RandBot (by D. Norman, 2003 [22, 24])

  Simply creates a random set of valid moves from the moves available to each unit and is in the DAIDE tournament as a reference [11].

- DumbBot (by D. Norman, 2003 [22, 24])

  Evaluates areas and creates orders to reach the highest ranked areas. The chances at moving towards an area is proportional to the evaluation of that area. Units that are already occupying the best area reachable try to support moves of other units [11].

- DiploBot (by F. McNeil, 2003 [24])

  Works much like DumbBot in that it bases its tactical decisions on the evaluation of areas. However, in DiploBot the area weights proceed from a more sophisticated stepped-iterative approach where a sequence of different modules modify the weights of each province based on some criteria. Based on the final weights a list of routes is generated for each unit, sorted by priority. DiploBot chooses the best possible combinations of routes for all units [11].

- Man'Chi (by B. Roberts, 2004 [23, 24])

  Based on a board analysis yielding goals on areas to conquer [14]. According to its creator its tactics are weak and its strategy horrible [23]. It seems to perform not too poorly against HaAI, though [11].

- HaAI (by F. Håård, 2004 [11, 24])

  HaAI is a multi-agent-system (MAS). One unit agent is created for each unit the bot controls. The unit agents evaluate their surroundings and create goals of movements which they submit to the MAS. The unit agents also declare their ability to support goals of other unit agents. The MAS might request the unit agents for additional goals or supports and ultimately decides the best combination of goals that can be met. HaAI was tested against other available bots (RandBot, DumbBot, DiploBot, and Man'Chi) in an open competition of DAIDE and shows to outperform competitors in score while being competitive in speed [11]. HaAI was implemented around the same time as the models conveyed in this work were.

- Project20M (by A. Huff et al, 2005 [14, 24])

  As DumbBot and DiploBot, order formation is based on area evaluations. Project20M uses two values of evaluation for each area: one for the bot having an army in that area and one for having a fleet there. Several iterations of an evaluation combination process produce a more general reflection of the map. Instead of praying that the search space is continuous, it is *created* leading gradually towards local optima. Based on this landscape, the best area each unit can conquer is chosen to move that unit to. No two units attack the same area and units support each other where needed. The best combination realizable is executed [14]. Project20M was presented right before delivery of this thesis.

Håård and Huff et al made very promising attempts in designing an automated Diplomacy player. One comment we do like to make is that it might be interesting to know how fast both bots beat other bots, measured in game seasons. It might not always be possible to play off the most eligible bots against each other, yet feasible to recreate another (RandBot, for instance, serving as a simple, but reliable reference). The most serious attempts that always beat RandBot, when given enough time (game seasons), might differ in their time of victory.

1.6 Scientific relevance for AI

Artificial Intelligence has been concerned with games for many years. Why games are so interesting lies in its perfect combination of problem complexity and world simplicity. Games provide for problems on the edge of computer capabilities, in a well-defined, conceivable world.

The most apparent aim of research in games is creating an entertaining opponent. People often tend to like playing against bots that mimic human properties. Also, solutions to game problems might inspire to apply the responsible techniques to other fields as well. Good strategic algorithms might for instance be extrapolated to skilled logistics systems. We might not even be concerned with the best possible solution, as long as the solution fits given requirements.

Inspiration on designing artificially intelligent agents could very well come from naturally intelligent systems like the human himself. Maybe even more interesting is the possible influence of an algorithm on how humans think they think.

In a way this work links up with a Master's thesis by Douma on the game of Happy Families [10]. Douma formalized knowledge representation in this game and tried to deduct reasonable decisions on what actions to take during the game.

1.7 Methodology

The adjudicator and the agent model are implemented in C++. The main reason for this is that C++ is an object oriented programming language which allows for efficient interaction between different objects. More specific: the agent would not be asked to resolve the board, but it is expected to know what resolution does. An agent in the game world could inquire the judge living in the same game world for resolution forecasts. Also, C++ programs execute very fast.

To maintain all data needed for and resulting from the simulations, a database has been set up in PostgreSQL. This allows for an organized, fast and reliable retrieval of the data, for the running models as well as for researcher. Moreover, it functions as an excellent backup system for the state of the program. Since the program continuously uses the latest data from the database and submits its latest results, a restart with minimal loss of data is possible. With simulation runs of several weeks the importance of this property should not be underestimated. Also, this manner of data maintenance allows for visualization on the internet. The database system can cope with simultaneous data streams from different processes.

The graphical user interface is implemented in PHP. It visualizes the simulated games and allows for statistical data analysis. The interface also reduces the gap to a complete game portal in which the agent could play against human players.

1.8 Overview

In chapter 2 the theoretical background of this work is described. The algorithms of both the judge model and the agent model are explained in chapter 3, followed by an outline of the simulations that were run with these models and their results in chapter 4. This thesis closes with a discussion of the achievements in this work, the conclusions we drew from them and some suggestions on future work on this subject in chapter 5.

2 Theory

Automation of reasoning starts with the choice on how to represent things. Humans may think in vague concepts and associations, machines cannot (yet). In this work, Diplomacy matters will be represented in *logic*.

How an automated agent should approach a game largely depends on how the game is made up. We need to know in what terms an agent should be thinking and producing. Again, we would rather see a formal description than a general idea, leading us to *game theory*.

The actual progress of artificial thoughts should be brought about by some kind of mechanism that considers alternatives. With the enormous solution space of Diplomacy, simple search mechanisms are not likely to accomplish much. As in most of these kinds of problems, the choice was made for *evolutionary computing*.

Most of the theoretical background on *logic*, *game theory*, and *evolutionary computing* has been borrowed from existing literature. However, some additions were made to build the bridge to Diplomacy. The following sections describe these in detail.

2.1 Logic

Diplomacy employs a strict notation of orders. In this work an extended, formal fact notation is required to make complex reasoning and deduction possible. Logic meets those needs perfectly. Also, the transcription of Diplomacy notation to logic can easily be made due to the systematic structure of both. The used notation links up with predicate logic in Van Benthem et al [3] and is specific to Diplomacy games.

2.1.1 Constants

Units are represented by the label *army* or *fleet*, depending on the unit's type. Units are denoted by u, v or w.

Areas are represented by a three letter code, according to the rulebook of Diplomacy [8]. For instance, *pru* refers to 'Prussia', *ber* refers to 'Berlin', and *sil* refers to 'Silesia'. Areas are denoted by a, b, c, or d.

Each of *austria*, *germany*, *turkey*, *italy*, *france*, *england*, and *russia* refers to its equally labeled empire. Since each player represents one empire and each empire is represented by one player, the two are formally ambiguous. In this work an agent is always a Diplomacy player and therefore often used in the same context too. Agents, players, and empires are denoted by i or j.

2.1.2 Order objects

In this thesis, an order is represented as an object, consisting of a label referring to the type of order and a few arguments. Four different classes of orders are distinguished: *holds*, *moves*, *supports*, and *convoys*. The first argument of an order describes the type of the unit for which the order is issued and the second refers to its current location. Depending on the type of order, the third argument may be void, refer to an area or refer to an entire order itself. Orders are denoted by X, Y or Z.

The order to *hold* the fleet in 'Prussia' is represented by: $hold(fleet, pru)$. In general, the order to hold unit u at area a:

$$hold(u, a)$$

Similar to a hold order, one could order the army in 'Berlin' to move to 'Silesia' with $move(army, ber, sil)$. In general, to move unit u from area a to area b:

$$move(u, a, b)$$

A support order can either support a hold order or a move order. For instance, the order $support(fleet, pru, hold(army, sil))$ denotes that the fleet in 'Prussia' supports the army in 'Silesia' to hold. Alternatively, $support(fleet, pru, move(army, ber, sil))$ would represent that the fleet in 'Prussia' supports an army move from 'Berlin' to 'Silesia'. One may *support* another *support* or *convoy* order to *hold*. That is, *supports* and *convoys* can only be *supported* to *hold*.

Let v be some unit and b and c be some areas. Then, in general, to support the order $X \in \{hold(v, b), move(v, b, c)\}$ with unit u in area a:

$$support(u, a, X)$$

Convoy orders have a structure, similar to that of support orders. To convoy an army from 'London' to 'Norway' with a fleet in the 'North Sea', one would issue $convoy(fleet, nth, move(army, lon, nwy))$. A convoy order thus refers to the order of the convoyed unit. Let order $X = move(v, b, c)$. Then, to convoy unit v from area b to area c with unit u in area a:

$$convoy(u, a, X)$$

2.1.3 Predicates

Properties of objects (represented by terms) are denoted by predicate symbols. Combining a particular predicate with one or more objects, we get an expression stating a certain property of those objects. Such a statement can be either *true* or *false*.

The fact that 'Germany' has an army in 'Berlin' would be noted as $at(germany, army, ber)$. In general, we could state that empire i has unit u in area a by writing:

$$at(i, u, a)$$

Another kind of board facts concern area ownership. The fact 'Germany owns Berlin' makes $own(germany, ber)$, and in general, empire i owns area a:

$$own(i, a)$$

In *Fall* seasons, area ownership might change. If an empire has a unit in a certain area, that area becomes property of that empire. So, in every *Fall*:

$$at(i, u, a) \longrightarrow own(i, a) \tag{1}$$

The area 'Berlin' contains a production center, which we write as $production(ber)$. The fact on a production center in any area a is denoted by:

$$production(a)$$

Now after every *Fall* the number of units each empire owns should not exceed the number of production centers that lie in that empire. In other words: the number of areas where an empire i has a unit should be at most the number of areas empire i owns that contain a production center:

$$|U_i| \leq |O_i \cap P| \tag{2}$$

with area sets U_i, O_i and P such that

$$a \in U_i \longleftrightarrow \exists u\ at(i, u, a)$$
$$a \in O_i \longleftrightarrow own(i, a)$$
$$a \in P \longleftrightarrow production(a)$$

The adjacency of any two areas depends not only on the location of both areas on the board, but also on the type of unit that attempts to move from one area to the other. For instance, armies cannot move to a sea area and fleets cannot move to some land areas. The adjacency of areas a and b for unit u, is denoted by:

$$adjacent(u, a, b)$$

We also combine constants and objects to state a relation between them. To denote that 'Germany' issues $move(army, ber, sil)$ we write: $issue(germany, move(army, ber, sil))$. In general, the issue of order X by empire i:

$$issue(i, X)$$

Whether or not an issued order is followed depends on other issued orders. Non-issued orders are never followed. The property whether an order X is followed or not is denoted by:

$$follow(X)$$

Convoyability is described with respect to an entire order. Let order X be the move of unit u from area a to area b, or $X = move(u, a, b)$. Then the convoyability of unit u from area a to area b is represented by the convoyability of X:

$$convoyable(X)$$

In some cases we might want to know if a unit is convoyable without using any path through a certain area (for instance because the fleet in that area was disrupted). Again assuming $X = move(u, a, b)$, the convoyability of unit u from area a to area b without using a path through area c is represented by:

$$convoyable\_dis(X, c)$$

Now we might describe order attainability in terms of adjacency and convoyability. The attainability of an order to move unit u from area a to area b is represented by:

$$attainable(X) \equiv adjacent(u, a, b) \vee convoyable(X) \tag{3}$$

And attainability of such an order, without crossing area c:

$$attainable\_dis(X, c) \equiv adjacent(u, a, b) \vee convoyable\_dis(X, c) \tag{4}$$

Additionally, we define two orders to be opposing when they are *moves* to each other's home area and both *moves* are feasible without *convoy*. Let $X = move(u, a, b)$ and $Y = move(v, b, a)$. Then:

$$\begin{aligned} opposing(X, Y) \equiv \; & [X = move(u, a, b) \wedge Y = move(v, b, a) \\ & \wedge adjacent(u, a, b) \wedge adjacent(v, b, a)] \end{aligned} \tag{5}$$

The predicate logic model of all objects, referring to terms in the language defined above is denoted by:

$$\mathcal{M}$$

If not all predicates in $\mathcal{M}$ are known, $\mathcal{M}$ is a partial model [2].

2.1.4 Functions

The strength of a particular order is found by counting the followed supports for that order and add one for the ordered unit itself. According to the rule book, if a unit u tries to dislodge another unit v, the strength of the move order for u does not include support orders issued by the same empire as the empire that unit v belongs to. Note that the omission of a support may depend on the success of a move order for unit v. That is, if unit v moves, it cannot be dislodged by u and thus no support should be omitted for contributing to own dislodgement. The strength of an order X is denoted by:

$$strength(X)$$

The resistance of a particular order is found in the same way as strength is, but now all followed supports are included. The resistance of a move order may only prevent other move orders to the same area. That is, for a move to some areas to be followed, the strength of that move must exceed the resistance of any other move to the same area. The resistance of an order X is denoted by:

$$resistance(X)$$

Suppose we have a set of sentences of predicates and logical operators in model $\mathcal{M}$. Then the sum Σ of that set is defined as the number of sentences in the set that are known to be *true* in $\mathcal{M}$. Model $\mathcal{M}$ might thus very well be partial, leaving unknown sentences out of the sum. In general, with logical sentences $\varphi_1, \varphi_2, \ldots, \varphi_{n+1}$:

$$\Sigma\{\varphi_1, \varphi_2, \ldots, \varphi_{n+1}\} = \begin{cases} 1 + \Sigma\{\varphi_1, \varphi_2, \ldots, \varphi_n\} & \text{if } \mathcal{M} \models \varphi_{n+1} \\ \Sigma\{\varphi_1, \varphi_2, \ldots, \varphi_n\} & \text{otherwise} \end{cases}$$

The sum of a set with one element is also denoted as the sum of that element:

$$\Sigma\varphi = \begin{cases} 1 & \text{if } \mathcal{M} \models \varphi \\ 0 & \text{otherwise} \end{cases}$$

Now, there is a relation between the *resistance* of an order and the sum of the set of all *supports* for that order. When the truth value of all sentences in $\mathcal{M}$ are known ($\mathcal{M}$ is complete), the *resistance* of an order X is equal to the sum of the set of successful supports for that order:

$$resistance(X) \equiv \Sigma R \tag{6}$$

with set R such that

$$\varphi \in R \longleftrightarrow \varphi = follow(support(u, a, X))$$

If X is not a *move* or X is a *move* to a vacant area, its *strength* is equal to its *resistance* since no unit could ever *support* X to dislodge anything. However, if X *is* a *move* to an area where empire i has a unit, the strength of X is equal to the sum of the set of followed supports for X from units belonging to empires other than empire i. Ergo, in the complete model $\mathcal{M}$:

$$strength(X) \equiv \Sigma S \tag{7}$$

with set S such that

$$\varphi \in S \longleftrightarrow \begin{cases} \varphi = follow(support(u, a, X)) & \text{if } X = hold(v, b) \\ \varphi = follow(support(u, a, X)) \wedge at(i, a) \wedge \neg at(i, c) & \text{if } X = move(v, b, c) \end{cases}$$

2.2 Game theory

As Binmore put it, game theory is mostly about what happens when people interact in a rational manner [4]. We should not only think of the games we play in our living room, but also of social matters like voting behavior, warfare, economics, and bargaining. Game theory finds its ways to applications in any of these fields, but in this work the road will lead to a game we play just for fun.

2.2.1 Game properties

Games can be divided by some very important properties. For a particular game, those properties greatly determine the way that game should be played, both by humans and by artificial agents [9].

Diplomacy is an infinite simultaneous multi-player zero-sum game with imperfect information. These game properties and their consequences are discussed below.

- Imperfect information

  Players in Diplomacy do not have access to all information about the game. They do see the current game state (the board), but they do not know what other players issue until resolution starts. Consequently, players somehow need to assume the unknown information. In Diplomacy one could figure that others issue orders that are likely to improve their position.

  A second kind of information Diplomacy players do not know regards negotiations. You never know what keen plans others discuss. Fortunately in *no-press* Diplomacy this is not a problem since no-one talks.

- Zero-sum

  Assume we have an evaluation function, indicating how well each player is doing at any time. The player's highest evaluation means an inevitable victory for that player (assuming he will play his best moves) and the lowest evaluation indicates he cannot avoid to lose. Since only one player can win a Diplomacy game, no two players' evaluations are the maximum. Even more, the sum of all players' evaluations should be the same at all times. If one player wins a certain amount, then the other players lose exactly the same amount. Strictly, the evaluations in a zero-sum game should sum to zero, but often different, constant sums are used to fit the problem at hand better. In this work, the sum of evaluations is always 1.

- Multi-player

  Diplomacy is a multi-player game, since more than one player take part in it. In fact, there are seven.

- Simultaneous

  Diplomacy is not a standard multi-player game. Players do not take turns, they all act simultaneously. This mainly accounts for the imperfectness of information described earlier. Furthermore, it thwarts evaluating one's position and seeing through strategies of others.

- Infinite

  A game is said to be infinite if the rules of that game allow players to play on forever. In Diplomacy, players could very well move their units to and fro or even not move

anything at all. Such behavior would never end in one player's owning 18 production centers. However, sometimes players beforehand agree upon a fixed number of rounds to play. This would make Diplomacy a finite game and most likely influences one's strategy towards the end. This work does not involve a limit on the number of rounds, known to the players, leaving Diplomacy an infinite game.

2.2.2 Game states

The game state describes the information that is available at a certain time. In some games (like Poker) this information differs for different players. All players do not necessarily know the same facts. What each individual player *does* know is called his local state.

A global state is the set of all available facts on what is known by whom [12]. In Poker it could describe that player 1 knows that the king of spades is in the hand of player 3 or that all players know that the seven of hearts is missing.

In some games (like Chess) both players always know the same things (what the board looks like and whose turn it is) and are therefore always in the same local state. The global state at a certain time in those games is the same as any local state at that time.

Right before players write their orders, the global state of a *no-press* Diplomacy game is also equal to any local state, since all information on the game at that time is available to anyone. We will define those global game states at those times.

A global state in this work concerns two things: the game round and the board configuration. Game rounds are represented by non-negative integers, referring to 'Spring 1901', 'Winter 1901', 'Spring 1902', 'Winter 1902', etc. respectively. In other symbols, round r is:

$$r \in \mathbf{N}_0$$

Formalizations of boards are conjunctions of at and own predicates, one for each unit location and area ownership, respectively. In general, if we have a board with m units on it and n owned areas, it would be represented by:

$$\beta = at(i_1, u_1, a_1) \wedge at(i_2, u_2, a_2) \wedge \cdots \wedge at(i_n, u_n, a_m) \wedge \\ own(i_{m+1}, a_{m+1}) \wedge own(i_{m+2}, a_{m+2}) \wedge \cdots own(i_{m+n}, a_{m+n})$$

Now, the global game state may be defined in terms of the game round and the configuration of the board. If a game were in round r with board configuration β, it's global state would be the pair:

$$\langle r, \beta \rangle$$

2.2.3 Actions

A local action is the action of one agent. In Diplomacy, one local action of an agent would include the issue of one order for each of that agent's units. For instance, if 'Germany' has a fleet in 'Kiel' and armies in 'Berlin' and 'Munich', this empire may perform a local action:

$$\begin{aligned} \lambda_{germany} = \quad & issue(germany, move(fleet, kie, den)) \\ \wedge \quad & issue(germany, move(army, ber, kie)) \\ \wedge \quad & issue(germany, move(army, mun, ruh)) \end{aligned}$$

Or in general, for agent i with n units:

$$\lambda_i = issue(i, X_1) \wedge issue(i, X_2) \wedge \cdots \wedge issue(i, X_n)$$

In many games (like Chess, Poker, Tic Tac Toe) agents perform local actions in turns, probably changing one ore more local states and if so, also the global state. In Diplomacy local actions have no effect unless all agents perform one at the same time. Although players do think for themselves, their orders are resolved as one event, one global action:

$$\alpha = \lambda_{austria} \wedge \lambda_{germany} \wedge \lambda_{turkey} \wedge \lambda_{italy} \wedge \lambda_{france} \wedge \lambda_{england} \wedge \lambda_{russia}$$

2.2.4 Search space

Let us assume that there are three possible board configurations β_0, β_1, and β_2 and the game lasts for three rounds. Figure 3 shows all twelve possible global states.

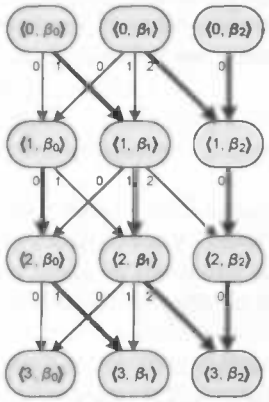

Figure 3: An example of a Diplomacy search space

In each global state there are a number of global actions that change the global state to another, represented by the branches from that node. Notice that nodes with the same board configuration β branch equally; the global action alternatives are independent of round r. Global actions are therefore denoted with respect to the board configuration only: α_β. The g^{th} branches from all nodes $\langle r, \beta \rangle$ with some r, represent global action α_β^g.

Ignore the different branch thickness for now: they will be explained in section 2.2.5.

2.2.5 Strategies

A pure local strategy describes the intention of one agent in all local game states he might encounter. In a game of imperfect information, like Diplomacy, they specify actions for each *information set* that agent might meet [4]. An information set describes all facts that are known by a certain agent.

If all possible game states would have a sequential number, pure strategies would be strings of labels, in which the gth element states the branch to take in state g. In Diplomacy, game states are defined in two-dimensional pairs. Although the alternatives to choose from are independent of the round, a particular board configuration may yield different intended global actions in different rounds. A pure Diplomacy strategy should specify the branch to pick for each board state in each round.

A pure global strategy specifies the intention of all players in all information sets. Imagine each player having a pure, local strategy of local actions for all game states he can think of.

Then their joint, global actions in all global states constitute a pure, global strategy (which not all players are necessarily aware of).

In the search space of the example in figure 3, all agents might come up with an action to make in each of the nine global states. Assume their plans form the global actions indicated by the thick branches in figure 3. Their global strategy would then be the 3×3 matrix:

$$s = \begin{Bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 1 & 2 & 0 \end{Bmatrix}$$

Each number in the array determines the branch to pick in a particular global state. For instance, the 1 on the bottom left of the array (at row 2, column 0) means that in global state $\langle 2, \beta_0 \rangle$ players intend to pick the first branch, making global action $\alpha^1_{\beta_0}$.

In general, if n board configurations exist and the game is known to last for m rounds. Then a pure global strategy is an $m \times n$ matrix, such that the branch that represents the planned action in global state $\langle r, \beta_p \rangle$ is:

$$s_{r,p}$$

Diplomacy has an unpleasantly enormous search space with n estimated over $5 \cdot 10^{165}$ and m infinite. Luckily, the aim of this work is not to find a pure, global strategy that covers each node. The analysis is focused on the situations an agent is expected to meet, rather than trying to foresee all and run out of time. A strategy that describes global actions for *some* global states is called a partial global strategy.

For instance, if the current global state is $\langle 0, \beta_0 \rangle$, we might choose not to worry about $\langle 0, \beta_1 \rangle$, and $\langle 0, \beta_2 \rangle$. If we also focus on the global states we expect to meet in the future, we might construct a partial global strategy for nodes $\langle 0, \beta_0 \rangle$, $\langle 1, \beta_1 \rangle$, and $\langle 2, \beta_1 \rangle$ only:

$$s = \begin{Bmatrix} 1 & \cdot & \cdot \\ \cdot & 1 & \cdot \\ \cdot & 2 & \cdot \end{Bmatrix}$$

2.3 Evolutionary computing

Human's way of thinking is not the only aspect we should crib from nature. And we did not. The sophisticated manner in which many forms of life were shaped to fit the properties of their world has inspired scientists to develop evolutionary computing. In analogy to the search for viable species by means of evolution, scientists search for solutions that fit problem properties by means of genetic algorithms.

Mitchell [21] explained that there are at least three (overlapping) meanings of *search*: search for stored data, search for paths to goals and search for solutions. The idea of the latter class is to efficiently find a solution to a problem in a large space of candidate solutions. These are the kind of search problems for which genetic algorithms are used.

In contradiction to the search for stored data, in a search for solutions each candidate is created while the search proceeds. A solution may be guessed or extensively adjusted over time, as long as it is possible to evaluate it. We mold our collection of hay pieces until one of them sufficiently looks like a needle.

Genetic algorithms work with a population of individuals. Metaphorically speaking, each individual holds DNA, consisting of chromosomes to represent one candidate solution. After the initiation of a population with individuals, two stages can be distinguished in the evolution process. First the best available individuals are selected from the population, and possibly

others are discarded. Second, the selected individuals are bred to create one or more new individuals. These two stages are repeated until certain criteria are met.

Mitchell [21] describes four central factors in the success of a genetic algorithm, as explained below.

2.3.1 Candidate encoding

Most genetic algorithm applications use fixed-length, fixed-order bit strings to encode candidate solutions. However, in recent years, there have been many experiments with other kinds of encodings. The most common encoding schemes can be classified in three groups:

1. Binary encodings
   Consisting of a fixed-length, fixed-ordered string of variables, each with two possible values.

2. Many-character and real-valued encodings
   As in binary encodings, the length and ordering of data in the encoding scheme is fixed. However, each variable contains a value out of a large or infinite set of possible values.

3. Tree encodings
   The length of the encoding is unlimited and the ordering represents tree structures.

Which encoding scheme to choose largely depends on the application at hand. Binary encodings allow for a higher degree of implicit parallelism [13] and heuristics about appropriate parameter settings have generally been developed in the context of binary encodings. However, binary encodings are unnatural for many problems and are prone to rather arbitrary orderings. A binary mapping of a continuous quantity might put originally distant candidates close together and vice versa. If solutions are likely to be found near the better individuals, many-character or real-valued encodings might be the better choice. Tree encodings allow for the representation of any tree, but could grow large in uncontrollable ways.

2.3.2 Selection methods

The second important factor in the success of genetic algorithms is the way individuals are selected for reproduction. Usually a small group of individuals (possibly just one or two) may reproduce the next generation whilst all others are discarded.

To push evolution forward it is wise to choose the good individuals over the bad ones. Often the top individuals are selected, but sometimes selection criteria are more subtle, including random selection based on fitness-proportionate chances.

The fate of individuals that are not selected should be decided as well. Those individuals could be expelled from the population to improve selection speed or kept alive to prevent regeneration of the same individuals later on.

Various selection methods are described in [21].

2.3.3 Genetic operators

Once individuals are selected, they reproduce. Two genetic operators are mostly used:

1. Crossover
   Two DNA strings of the selected individuals are cut into pieces from which one or more new individuals are constructed. Many varieties exist on where to cut and how often.

2. Mutation
   One or more chromosomes in the DNA of a selected individual are randomly changed to form new DNA, a new individual.

Many other genetic operators exist, as well as combinations of several. The best choice again depends on the application at hand. Crossover might be desirable when two partial solutions are likely to form a better solution together. Crossovers allow for big jumps in the search space, possibly breaking out of local optima in the search space. Mutations may do better when the search space is continuous with few local optima.

2.3.4 Parameter settings

The fourth decision to make in implementing a genetic algorithm is how to set the values for the various parameters, such as population size, crossover rate, and mutation rate. In literature, there is too much discussion on setting and adapting parameters to survey here. In general, the performance of the algorithm with various parameter settings is decisive in either manual or automatic adaptation of parameters. The optimal parameter values are often not permanently chosen, since they might be discarded on the grounds of inefficiency during simulation.

3 Design

The model designs in this work came about by combining logic, game theory, and evolutionary computing and were implemented in C++. They make use of a PostgreSQL data base to manage all necessary data. The following sections elaborate on the detailed design choices.

3.1 Data structure

To maintain all necessary data for a game of Diplomacy, a data structure is set up in a PostgreSQL database. One of the main reasons for this choice is the high flexibility of a query language. Any selection and any ordering is easily feasible. A second important reason to use PostgreSQL is its great accessibility: the database can be accessed by various kinds of processes, written in different programming languages.

The database contains the following data classes:

1. *User*
   The class *User* describes agents, either human or artificial. Most attributes of this class describe personal data of the user. Other attributes keep track of statistical data on that particular user. Instances of *User* are scientifically irrelevant.

2. *Area*
   In the class *Area* all board specifications are stored in terms of area characteristics. Some of the attributes hold information about physical aspects of areas (like their absolute position), which is used to generate a visual representation of those areas. Others describe the limitations the board inflicts on the game, like area types or which areas are adjacent for each unit. This information is needed to adjudicate game actions.

3. *Game*
   Whenever a new game is started, an instance of *Game* is created, describing what type of game it is, who plays which position, what phase it is in, how long each phase lasts, when the next deadline is, and a few statistical values.

4. *Board*
   Every resolution of every game produces an instance of the class *Board*, which, in addition to the appropriate game name and phase, describes the board in terms of area ownership and unit locations. This information is needed to generate a visual representation of the board, to determine the possibilities an agent has and to resolve following boards.

5. *Order*
   Instances of class *Order* hold logical representations of issued orders and information on the game, phase, and empire they belongs to. Some order aspects (like the areas involved) are redundantly included to improve information extraction speeds.

All information in the data base is dynamic in the sense that it may be changed by an agent or a process while being read by another. Since the number of games may be high and the processing time to resolve them long, it is desirable to allow new input from agents at the same time. A full listing of all data classes is given in appendix A.

3.2 Automated adjudicator model

The first design of the adjudication model was condition-based. The model was built to decide on the success of orders as soon as they emerged. If a decision depended on a condition, an attempt was made to decide on the outcome of that condition first. Recursive dependency processing led to the fixable conditions. In appendix B an example is described in which diagram 29 from the Diplomacy rulebook [8] is adjudicated.

The condition-based judge was able to solve some circular dependencies by assuming decisions never to rely on themselves. Whenever a decision was processed, it was temporarily assumed successful, so that it could never prevent its own success. However, with the growing complexity of the test cases, the difficulty of recognizing the problem (if there was one) grew as well. At some point, in particular when test cases became paradoxical, the manner in which to approach the adjudication could not be recognized from the local condition structure. It might have been possible to create a mechanism that would foresee paradoxes by the structure of the entire order set. Since such a mechanism would greatly slow the adjudication of the simpler case it was not heartily welcomed.

The reader might remark that the problems with a condition-based design were largely proclaimed in section 1.4.3. Unfortunately it was not until this design was implemented and tested that those problems became apparent. The findings with the first judge have led to the choice to re-design it in a logic-based manner.

The final design of the logic-based adjudication model is named the Automated Truthful Logical Adjudication System (Atlas). When given a particular board state and a set of issued orders, Atlas goes through a number of stages to determine whether or not each order is followed, as shown in figure 4.

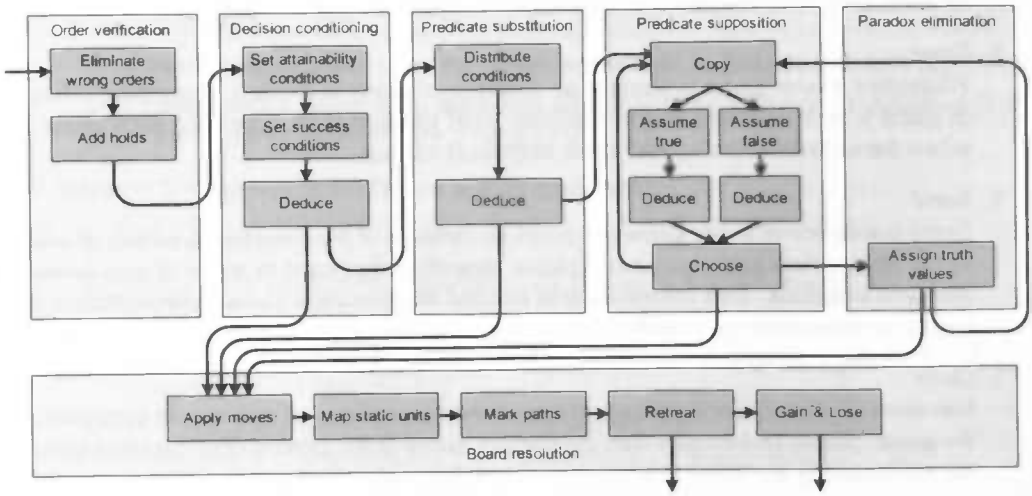

Figure 4: Schematic representation of the logic-based adjudication model 'Atlas'

Atlas starts at the top left with order verification, continuing along the arrows shown. Possibly, Atlas need not complete all stages to solve the problem. Whenever all decisions are made (at the earliest after the decision conditioning stage) Atlas skips to the board resolution stage and determines the new board state.

The next sections clarify how the design of each stage is filled in, including the simple adjudication example below (test case 6.G.1 from [17]).

Units on the board

- England: army Norway; fleet Skagerrak
- Russia: army Sweden

Figure 5: Test case 6.G.1

Formal representation of orders

- England:
  $issue(england, move(army, nwy, swe))$
  $issue(england, convoy(fleet, ska, move(army, nwy, swe)))$
- Russia:
  $issue(russia, move(army, swe, nwy))$

A fairly complex adjudication example requiring Atlas to perform all stages can be found in appendix C.

3.2.1 Order verification

First, Atlas filters out illegal and invalid order issues. An issue is *illegal* when the ordered unit does not exist or does not belong to the issuing empire. Illegal order issues will be regarded as if no empire ever issued them at all.

Invalid orders regard *supports* of orders that have not *legally* been issued. Those supports are not removed from the set of issued orders, but marked *invalid* so that they are ignored in the remaining process, forcing those orders to fail. This design detail is a direct result from resolution rule 9 from section 1.2.3.

Note that an *invalid* order might be *legal*. Its *support* just serves no purpose.

Secondly, all units that have not (or illegally) been issued receive a hold order, according to the game regulations, stating that holds need not explicitly be specified [8]. Any added hold order will be regarded as if it was actually issued by the owning empire. Now all remaining orders are legal and $issue(i, X)$ is *true* if and only if empire i legally issued order X.

The proposed example does not include wrong orders.

3.2.2 Decision conditioning

For each legally issued order a decision is to be made whether or not that order is followed. In other words, we want to know the truth value of the predicate $follow(X)$ for each order X for which $issue(i, X)$ is *true* for some i. Atlas tries to find the conditions that the success of each issued order relies upon. Thus, for each issued order X, Atlas generates a sentence that describes when that order is actually followed:

$$\forall X \, [\, follow(X) \longleftrightarrow \exists i \, (issue(i, X) \wedge \sigma) \,] \tag{8}$$

where σ is a conjunction of sentences, specific for the class of order X. For each order class a distinct group of rules is used to determine the content of σ. For supports there are two groups: one for the support of a *hold* and one for the support of a *move*.

The resolution rules from the rulebook mostly describe what *thwarts* the success of orders [8]. Atlas, on the other hand, bases its decisions on what is *required* for order successes. Its condition rules are therefore no one-on-one mapping from the rules in sections 1.2.3 and 1.2.4. They do agree with them, though.

The italic numbers behind the condition rules below refer to the resolution rules they relate to. In each condition rule, X is the order issued by empire i. Orders X and Y are never equal, since the outcome of a decision never directly depends on itself.

- Hold

  1. Any foreign move to the home area should not be followed *(2, 4)*.

  Assume $X = hold(u, a)$. Then σ equals:

  1. $\forall v, b, Y, j\ [Y = move(v, b, a) \wedge j \neq i \wedge issue(j, Y)$
     $\rightarrow \neg follow(Y)]$

- Move

  1. The move should be attainable.
  2. There should not be a friendly unit staying in the target area *(2, 5, 12)*.
  3. A foreign unit staying in the target area should be defeated *(2, 3, 5)*.
  4. Any other move to the target area should be defeated, unattainable or dislodged by a unit coming from the target area *(2, 3, 10, 11, 18)*.
  5. An opposite, friendly move should be avoided by convoy and should be followed *(2, 5, 6, 12, 19)*.
  6. An opposite, foreign move should be defeated or avoided by convoy. In the latter case the foreign move should be followed or defeated *(2, 3, 5, 6, 19)*. Note that a move order can never receive support to hold and thus has strength of 1 when it is not followed.
  7. A friendly move, away from the target area should be followed *(2, 5, 7, 12)*.
  8. A foreign move, away from the target area should be followed or defeated *(2, 3, 5, 7)*. Again the strength of such a failing move is 1.

  Assume $X = move(u, a, b)$. Then σ is a conjunction of:

  1. $attainable(X)$
  2. $\forall v, Y, Z\ [Y \in \{hold(v, b), support(v, b, Z), convoy(v, b, Z)\} \wedge issue(i, Y)$
     $\rightarrow \bot]$
  3. $\forall v, Y, Z, j\ [Y \in \{hold(v, b), support(v, b, Z), convoy(v, b, Z)\} \wedge j \neq i \wedge issue(j, Y)$
     $\rightarrow strength(X) > strength(Y)]$
  4. $\forall v, w, c, Y, j\ [c \neq a \wedge Y = move(v, c, b) \wedge issue(j, Y)$
     $\rightarrow strength(X) > resistance(Y) \vee \neg attainable(Y) \vee follow(move(w, b, c))]$
  5. $\forall v, Y\ [Y = move(v, b, a) \wedge issue(i, Y) \wedge opposing(X, Y)$
     $\rightarrow [convoyable(X) \vee convoyable(Y)] \wedge follow(Y)]$
  6. $\forall v, Y, j\ [Y = move(v, b, a) \wedge issue(j, Y) \wedge j \neq i \wedge opposing(X, Y)$
     $\rightarrow strength(X) > strength(Y)$
     $\vee((convoyable(X) \vee convoyable(Y)) \wedge (follow(Y) \vee strength(X) > 1))]$
  7. $\forall v, c, Y\ [Y = move(v, b, c) \wedge issue(i, Y) \wedge \neg opposing(X, Y)$
     $\rightarrow follow(Y)]$
  8. $\forall v, c, Y, j\ [Y = move(v, b, c) \wedge j \neq i \wedge issue(j, Y) \wedge \neg opposing(X, Y)$
     $\rightarrow follow(Y) \vee strength(X) > 1]$

- Support of a hold

  1. Any foreign move to the home area should be unattainable *(2, 13, 15)*.

  Assume $X = support(u, a, hold(v, b))$. Then σ equals:

  1. $\forall w, c, Y, j\ [Y = move(w, c, a) \wedge j \neq i \wedge issue(j, Y)$
     $\rightarrow \neg attainable(Y)]$

- Support of a move

  1. A foreign move from the target area of the supported move to the home area should not be followed *(2, 13, 14, 16)*.
  2. Any foreign move to the home area should not be followed and not be attainable if the target area of the supported move is to be avoided *(2, 13, 14, 15, 16, 21, 22)*.
  3. There should not be a friendly unit staying in the target area of the supported move *(2, 12)*.
  4. A friendly move, away from the target area of the supported move should be followed *(2, 12)*.

  Assume $X = support(u, a, move(v, b, c))$. Then σ is a conjunction of:

  1. $\forall w, Y, j\ [Y = move(w, c, a) \wedge j \neq i \wedge issue(j, Y)$
     $\rightarrow \neg follow(Y)]$
  2. $\forall w, d, Y, j\ [d \neq c \wedge Y = move(w, d, a) \wedge j \neq i \wedge issue(j, Y)$
     $\rightarrow \neg follow(Y) \wedge \neg attainable\_dis(Y, c)]$
  3. $\forall w, Y, Z\ [Y \in \{hold(w, c), support(w, c, Z), convoy(w, c, Z)\} \wedge issue(i, Y)$
     $\rightarrow \bot]$
  4. $\forall w, d, Y\ [Y = move(w, c, d) \wedge issue(i, Y)$
     $\rightarrow follow(Y)]$

Notice that the second condition to the success of a *support of a move* does not incorporate rule *23*. The support should be followed if its success would make the support impossible to cut. Unfortunately, at the decision level it is impossible to foresee the consequences of a decision before we make it. Rule *23* is to be applied at a later stage, when we know it fits. For now it is important to see that supports that cannot be decided yet are preserved to be caught in later stages.

- Convoy

  1. Any foreign move to the home area should not be followed *(2, 4, 17)*.

  Assume $X = convoy(u, a, move(v, b, c))$. Then σ equals:

  1. $\forall w, d, Y, j\ [Y = move(w, d, a) \wedge j \neq i \wedge issue(j, Y)$
     $\rightarrow \neg follow(Y)]$

Each issued order is treated just once to determine its success conditions from the appropriate group. However, Atlas does not blindly add entire *material conditionals* ($\varphi \rightarrow \psi$) to the conjunction σ. All *antecedents* (φ) contain *issue* predicates, so Atlas tries to match them with all other issued orders. Whenever a match is found, the *consequent* (ψ) is added to σ, according to equation 9; knowing that φ is *true* and $\varphi \rightarrow \psi$ *should* be *true*, Atlas adds ψ to σ.

$$\varphi \wedge (\varphi \rightarrow \psi) \Longrightarrow \psi \tag{9}$$

The conditions to the success of *hold* orders are not actually determined, since no other order success could depend on them. Note that order successes might depend on the *existence* of issued hold orders, though. It is far more efficient to determine *hold* order successes when all *move* order successes are known. So for now, *hold* orders are ignored.

The conditions that predicates *convoyable*, *attainable*, *attainable_dis*, and *opposing* rely upon are determined right away, leaving those predicates out of σ. A move is *attainable* when the involved areas are *adjacent* for the involved unit or when that move is *convoyable*, if required without crossing a particular area (equations 3 and 4). Two orders are *opposing* when they are moves to each other's home area and those areas are adjacent for both units involved (equation 5).

All remaining *convoyable* predicates are expressed in terms of successes of convoys that form a path between the two areas: a move is *convoyable* if and only if all convoys in at least one useful path are followed. The *adjacent* predicates Atlas is left with are known for any two areas and unit types.

According to equations 6 and 7, *strength* and *resistance* functions should end up equal to the sum of some set of sentences. For each function, Atlas keeps such a set. Each time a support decision is visited, its conditions form an additional element in the sets that belong to *strength* and *resistance* functions that match that support. In the case of a *strength*, the entire condition of the support is included in the new argument; in the case of a *resistance*, only conditions generated by hold support rule 1 and move support rules 1 and 2 are included. Such wider conditions include supports of own dislodgement, since move support rule 3 and 4 forbid exactly that. Ultimately (when all predicates and sentences are known), the sum of each set of sentences should determine the value of the function it belongs to.

Atlas treats all move orders before any support order because of two reasons. The first reason is that support order conditions might contain attainability predicates of move orders (hold support rule 1). Since the conditions of those predicates are already determined by move order rule 1, Atlas can replace *attainability*(*_dis*) predicates immediately with the corresponding conditions.

The second reason involves the calculation of *strength* and *resistance* functions. Atlas could, for each occurrence of a function, search the order list for matching *supports* to fill its set of sentences. Alternatively, Atlas could, for each *support*, add the condition of its success to the appropriate sets. Since the move rules imply an average of almost two functions per conflicting order, whereas an average of no more than one support is expected for an order, the second method is implemented for its presumably higher efficiency. Now, this distribution of supports is only possible after the inclusion of all necessary functions in the success conditions of all *move* orders.

In this manner, Atlas determines all order success conditions without occurrences of the predicates *convoyable*, *attainable*, *attainable_dis*, and *opposing*. Also, all *strength* and *resistance* variables are (conditionally) determined by sums of sentence sets. At this stage all order decision conditions are defined solely in terms of truth values and the successes of move and convoy orders (each σ is a conjunction of *trues*, *falses*, and *follow* predicates of *move* and *convoy* orders). The structure of a condition may be rather complex, though, including conjunctions, disjunctions, negations, and sentence set sum comparisons.

For the orders of test case 6.G.1, the decision condition rules imply the sentences below. The conditions to the success of both move orders are a result of move order decision conditioning rules 1 and 6. No threat is found to the success of the convoy order.

1. $follow(move(army, nwy, swe)) \longleftrightarrow attainable(move(army, nwy, swe)) \wedge$
   $[strength(move(army, nwy, swe)) > strength(move(army, swe, nwy)) \vee$
   $((convoyable(move(army, nwy, swe)) \vee convoyable(move(army, swe, nwy))) \wedge$
   $(follow(move(army, swe, nwy)) \vee strength(move(army, nwy, swe)) > 1))]$

2. $follow(convoy(fleet, ska, move(army, nwy, swe))) \longleftrightarrow true$

3. $follow(move(army, swe, nwy)) \longleftrightarrow attainable(move(army, swe, nwy)) \wedge$
   $[strength(move(army, nwy, swe)) > strength(move(army, swe, nwy)) \vee$
   $((convoyable(move(army, swe, nwy)) \vee convoyable(move(army, nwy, swe))) \wedge$
   $(follow(move(army, nwy, swe)) \vee strength(move(army, swe, nwy)) > 1))]$

Ares determines the conditions to *attainable* predicates right away. In this case, they are both *true*, since 'Norway' and 'Sweden' are adjacent from both units' respect (at this point the opposition of the armies is irrelevant). The *convoyable* predicates rely on the success of the convoy orders that form an appropriate path from the current to target area of the involved move order. In this example the convoyability of the move from 'Norway' to 'Sweden' depends on the success of the convoy order. The opposite move is not convoyable. Since no supports are issued, all *strength* functions are noted zero. The result is below.

1. $follow(move(army, nwy, swe)) \longleftrightarrow true \wedge$
   $[0 > 0 \vee ((follow(convoy(fleet, ska, move(army, nwy, swe))) \vee false) \wedge$
   $(follow(move(army, swe, nwy)) \vee 0 > 1))]$

2. $follow(convoy(fleet, ska, move(army, nwy, swe))) \longleftrightarrow true$

3. $follow(move(army, swe, nwy)) \longleftrightarrow true \wedge$
   $[0 > 0 \vee ((false \vee follow(convoy(fleet, ska, move(army, nwy, swe)))) \wedge$
   $(follow(move(army, nwy, swe)) \vee 0 > 1))]$

3.2.3 Logical deduction

When all conditions have been set, Atlas will simplify them according to logical deduction rules 10 through 19. Atlas visits each (sub)sentence in each condition σ recursively; unknown sub sentences are deduced before an entire sentence is. Whenever a sentence fits a term on the left side of an equation, it is replaced by the term on the right side.

Greek symbols $\varphi_1, \ldots, \varphi_n$, and $\psi_1, \ldots, \psi_m$ denote sentences. A sentence might be a predicate, a conjunction of sentences, a disjunction of sentences or a comparison of sentence set sums.

- Conjunctions
  Any conjunction containing a *false* is replaced with the truth value *false* (equation 11) and any *true* is removed (equation 12). When all sentences but one are *true* and thus removed, the remaining conjunction of just one sentence (it is still marked as a conjunction) is replaced with that sentence (equation 10).

$$\varphi \wedge \emptyset \Longleftrightarrow \varphi \tag{10}$$

$$\varphi_1 \wedge \cdots \wedge \varphi_n \wedge false \Longleftrightarrow false \tag{11}$$

$$\varphi_1 \wedge \cdots \wedge \varphi_n \wedge true \Longleftrightarrow \varphi_1 \wedge \cdots \wedge \varphi_n \tag{12}$$

- Disjunctions
  Disjunctions are handled in much the same way as conjunctions are. Now the disjunction is replaced with a *true* if one of the sentences in the conjunction is known to be *true* (equation 14) and occurrences of *false* are removed (equation 15). Again, when only one sentence remains, the disjunction is replaced with that sentence (equation 13).

$$\varphi \vee \emptyset \iff \varphi \tag{13}$$

$$\varphi_1 \vee \cdots \vee \varphi_n \vee \mathit{true} \iff \mathit{true} \tag{14}$$

$$\varphi_1 \vee \cdots \vee \varphi_n \vee \mathit{false} \iff \varphi_1 \vee \cdots \vee \varphi_n \tag{15}$$

- Negations
  Negations of truth values are replaced with their opposite truth values (equation 16, 17) and double negations are removed (equation 18).

$$\neg \mathit{true} \iff \mathit{false} \tag{16}$$

$$\neg \mathit{false} \iff \mathit{true} \tag{17}$$

$$\neg\neg\varphi \iff \varphi \tag{18}$$

- Set sums
  Occurrences of *false* in a set do not influence the sum of that set, so we might as well remove them (equation 19).

$$\Sigma\{\varphi_1, \ldots, \varphi_n, \mathit{false}\} \iff \Sigma\{\varphi_1, \ldots, \varphi_n\} \tag{19}$$

- Comparisons
  Equal terms on each side of a comparison may be omitted. In this case, Atlas removes *trues* when they appear on both sides of the comparison (equation 23). (Equation 19 already removes *falses* on either side.) The sum of a set S gradually increases as more sentences in S become known to be *true* while the number of sentences in that set might decrease (due to equations 19 and 23). Notice that ΣS will never exceed $|S|$ and that they become equal when all predicates have a definite truth value.

  Sometimes it is possible to deduce the truth value of a comparison before all sentences in both sets are known. Equations 20 through 22 describe the deduction rules for those cases. If there are not enough sentences in the set on the left of the comparison to ever exceed the current sum on the right, the comparison fails. Likewise, when the sum on the left has outgrown the number of sentences in the set on the right, the comparison is bound to succeed (equation 20). Whenever all sentences in the set on the right have been eliminated, only one *true* on the left is enough to justify the comparison (equation 21). Even so, could the sum of one *true* only exceed sets of *falsified* sentences (equation 22).

$$\Sigma S > \Sigma T \iff \begin{cases} \mathit{false} & \text{if } |S| \leq \Sigma T \\ \mathit{true} & \text{if } \Sigma S > |T| \\ \Sigma S > \Sigma T & \text{otherwise} \end{cases} \tag{20}$$

$$\Sigma\{\varphi_1, \ldots, \varphi_n\} > \Sigma\emptyset \iff \varphi_1 \vee \cdots \vee \varphi_n \tag{21}$$

$$\Sigma \mathit{true} > \Sigma\{\varphi_1, \ldots, \varphi_n\} \iff \neg\{\varphi_1 \vee \cdots \vee \varphi_n\} \tag{22}$$

$$\Sigma\{\varphi_1, \ldots, \varphi_n, \mathit{true}\} > \Sigma\{\psi_1, \ldots, \psi_m, \mathit{true}\} \iff \Sigma\{\varphi_1, \ldots, \varphi_n\} > \Sigma\{\psi_1, \ldots, \psi_m\} \tag{23}$$

The structure of the condition rules in section 3.2.2 ensures that additional decisions are only made when Atlas needs to. A conditional decision is encapsulated in the decision conditioning rules and eliminated by deduction when paltry.

By the described deduction, Atlas greatly reduces the conditions of each order success decision. After deduction, if all successes are determined (all conditions are either *true* or *false*), Atlas skips to the board resolution stage.

In the case of our example, the success sentences would get reduced to:

1. $follow(move(army, nwy, swe)) \longleftrightarrow$
   $follow(convoy(fleet, ska, move(army, nwy, swe))) \wedge follow(move(army, swe, nwy))$
2. $follow(convoy(fleet, ska, move(army, nwy, swe))) \longleftrightarrow true$
3. $follow(move(army, swe, nwy)) \longleftrightarrow$
   $follow(convoy(fleet, ska, move(army, nwy, swe))) \wedge follow(move(army, nwy, swe))$

3.2.4 Predicate substitution

Some decisions must depend on others to reach this stage. Remember that the only unknown terms in decision conditions are *follow* predicates. Resolving continues with a sequential substitution of those predicates with the conditions *they* rely upon (equation 24).

$$[\varphi \leftrightarrow \psi_1 \wedge \cdots \wedge \psi_{n+1}] \wedge [\psi_{n+1} \leftrightarrow \chi_1 \wedge \cdots \wedge \chi_m] \Longrightarrow [\varphi \leftrightarrow \psi_1 \wedge \cdots \wedge \psi_n \wedge \chi_1 \wedge \cdots \wedge \chi_m] \quad (24)$$

When a sentence φ depends on a conjunction of sentences of which one (ψ_{n+1}) depends on another conjunction of sentences $\chi_1 \wedge \cdots \wedge \chi_m$, Atlas replaces ψ_{n+1} with $\chi_1 \wedge \cdots \wedge \chi_m$.

This way, the truth values of decisions that (indirectly) depend on known predicates are determined. Each predicate is substituted just once, avoiding eternal loops when they indirectly depend on themselves.

Again, logical deductions are applied wherever possible and Atlas checks if all conditions are satisfied. If so, the model skips to board resolution.

Continuing our example, any predicate $follow(convoy(fleet, ska, move(army, nwy, swe)))$ on the right side of the arrow is replaced by *true*. Thereafter is each $follow(move(army, swe, nwy))$ predicate on the right replaced by $true \wedge follow(move(army, nwy, swe))$. After logical deductions the sentences below remain.

1. $follow(move(army, nwy, swe)) \longleftrightarrow follow(move(army, nwy, swe))$
2. $follow(convoy(fleet, ska, move(army, nwy, swe))) \longleftrightarrow true$
3. $follow(move(army, swe, nwy)) \longleftrightarrow follow(move(army, nwy, swe))$

3.2.5 Predicate supposition

At this point circular dependencies must exist. One or more decisions must (indirectly) depend on themselves.

To find the outcome of a self-dependent set of orders, Atlas assumes the first occurring *follow* predicate to be *true* and deduces the remaining ones by substitution and deduction. Then the same predicate is assumed *false* and again the successes of other orders are determined. Now, the predicate becomes that truth value that induces the success of supports that would fail otherwise (in compliance with resolution rule 23 from section 1.2.4):

$$\varphi = \begin{cases} true & \text{if } \exists u, a, X\ [\varphi \rightarrow follow(support(u, a, X)) \wedge \neg\varphi \rightarrow \neg follow(support(u, a, X))] \\ false & \text{if } \exists u, a, X\ [\varphi \rightarrow \neg follow(support(u, a, X)) \wedge \neg\varphi \rightarrow follow(support(u, a, X))] \\ true & \text{otherwise} \end{cases} \tag{25}$$

If a support exists whose success depends on a particular truth value assignment to some predicate, that assignment should be made. If no such support exists, the orders involve a circular movement, which should succeed. In this case φ is the *follow* predicate of one of the circular moves and should be *true* (resolution rule 7, section 1.2.3).

Logic fails when equation 25 suggests the assignment of both *true* and *false* to a predicate. This happens when the success of one support requires a move to succeed, while the success of another support requires the same move to fail. Atlas then applies paradox elimination to the decisions involved.

If more circular dependencies exist (when the supposition did not make *all* decisions) Atlas continues predicate supposition for yet unknown predicates.

This stage solves the adjudication of our example. The only unknown predicate on the right of the arrow in any sentence is $follow(move(army, nwy, swe))$. If we assume the predicate to be *true* we get:

1. $follow(move(army, nwy, swe)) \longleftrightarrow true$
2. $follow(convoy(fleet, ska, move(army, nwy, swe))) \longleftrightarrow true$
3. $follow(move(army, swe, nwy)) \longleftrightarrow true$

Assuming the same predicate to be *false* yields:

1. $follow(move(army, nwy, swe)) \longleftrightarrow false$
2. $follow(convoy(fleet, ska, move(army, nwy, swe))) \longleftrightarrow true$
3. $follow(move(army, swe, nwy)) \longleftrightarrow false$

Ares chooses the predicate $follow(move(army, nwy, swe))$ to be *true* since no support would fail due to this assignment. All orders succeed.

3.2.6 Paradox elimination

The set of orders is paradoxical when the assignment of *true* to some predicate favors the success of a support over another, whereas *false* produces the opposite. Applying rule 23 to one support would violate the same rule for the other.

The solution is simple. Conform resolution rule 24 (section 1.2.4), all dependent move orders fail and all dependent support and convoy orders succeed. Those support and convoy orders have nonetheless become useless in this case since no unit actually succeeds in moving and moves are the only orders that can actually change the board configuration.

If more circular dependencies exist, Atlas continues the supposition of truth values for unknown predicates (possibly involving more paradox eliminations). If not, board resolution is next.

Our example 6.G.1. does not involve a paradox, since it was solved by the predicate supposition stage. The adjudication example of test case 6.F.23 in appendix C *does* require paradox elimination.

3.2.7 Board resolution

The only thing that can actually change the board is a *move*. Atlas thus simply needs to move those units that received a move order that was successfully followed. To avoid a unit to dislodge a unit that was supposed to move itself, Atlas maps all units to a new, blank board. First all successful moves are mapped to the appropriate areas on the new board and then all static units are copied.

The resolution of test case 6.G.1 is that the units in 'Norway' and 'Sweden' exchange places.

If the current season is *Fall* (round r is even), all land areas containing a unit become property of that unit's empire (equation 1). The ownership of areas with no unit remains the same.

As far as Diplomacy rules concern, the new board is now made and resolution is done. The new global state is the pair of the previous round plus one and the new board configuration. However, this research asks for a few operations more.

As will be explained in section 3.5, the path of convoys is needed to draw their graphical representation. Atlas marks each convoy path with flags on each visited area, pointing to the next area in that path.

Retreats, disbands, and builds form a favor Atlas does the agents. The specifications of the agent model do not include these actions, but they are required for the progression of the game. Atlas makes a very poor attempt to retreat: the model does not. Dislodged units are simply disbanded (removed from the board). If the season is *Spring*, Atlas stops. If the season is *Fall*, units are randomly disbanded or built to fit the number of production centers each empire owns (equation 2).

3.3 The agent model

An agent model was created and named Agent of Rationally Evolving Strategies (Ares). When presented with a board situation, Ares plans actions for the game states it is expected to meet by means of genetic algorithms.

Although Ares is in control of its own actions only, it needs to be prepared for actions of opponents. It does so by considering actions for all agents (global actions). The only agent distinction Ares makes is between himself (agent 0) and its opponents (agents 1 through 6). A global action α contains one own local action λ_0 and six local actions $\lambda_1, \ldots, \lambda_6$ for the opponents, we call the *complement* of λ_0, or λ_0:

$$\alpha = \lambda_0 \wedge \underbrace{\lambda_1 \wedge \lambda_2 \wedge \cdots \wedge \lambda_6}_{\lambda_0}$$

The planning of global actions makes up global strategies. The quest for a good strategy is slightly more complicated than searching for a global strategy that suits Ares. Such a strategy might involve actions of other agents that benefit Ares, which sounds more like wishful thinking than rational thinking. Optimization should not depend on these uncontrollable matters, but they should not be ignored either. Ares assumes the worst-case scenario in which opponents gang up on him. For each own local action λ_0 in considered strategies, it tries to find complementary actions λ_0 that suit the model worst.

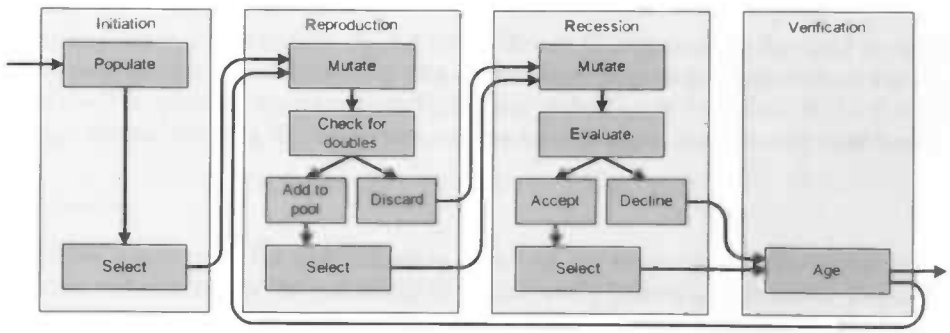

Figure 6: Schematic representation of the strategy evolution model 'Ares'

Ares pursues a mechanism as shown in figure 6. A population of individuals (series of global actions) is constantly evolved by *reproducing* new individuals from the best ones available. New individuals only differ in the local actions they contain for Ares (λ_0). At the same time *recession* of the best individuals available accounts for changes in the complementary local actions for opponents (λ_0). Note that *recession* does *not* produce new individuals (as *reproduction* does); it worsens the existing individuals.

The reproduction and the recession algorithm work with one global action in an individual (one limb) at a time, as explained further on. Evolution iteratively *reproduces* the *selected* limbs that endure the *recessions* until all limbs in the *selected* individual have reached a certain *age*. The number of *recession* cycles determines the age of a limb. In the end, the best individual delivers one series of global actions from which Ares extracts its next local action.

The next section describes the exact internal representation Ares copes with, followed by a detailed description of the mechanism components.

3.3.1 Internal representation

Series of global actions are the key elements in Ares' reasoning algorithm. Each unique series the algorithm can compose with the available global actions at a time is a unique individual. Internal representations are not that concrete, though: global actions are distributed over several stored lists. The text below explains how.

Given a particular board configuration β, the global actions branching from nodes $\langle r, \beta \rangle$ are equal for all r. Therefore, for each *board*, Ares keeps track of global action alternatives, as shown in the upper half of figure 7.

board	β_0	β_1	$\cdots$	β_n
actions	$\alpha^0_{\beta_0}$	$\alpha^0_{\beta_1}$	$\cdots$	$\alpha^0_{\beta_n}$
	$\alpha^1_{\beta_0}$	$\alpha^1_{\beta_1}$	$\cdots$	$\alpha^1_{\beta_n}$
	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$r = 0$	$s_{0,0}$	$s_{0,1}$	$\cdots$	$s_{0,n}$
$r = 1$	$s_{1,0}$	$s_{1,1}$	$\cdots$	$s_{1,n}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$r = m$	$s_{m,0}$	$s_{m,1}$	$\cdots$	$s_{m,n}$

Figure 7: Internal data representation

The stored global actions form the body parts of the individuals in the population: each individual is a series of global actions, starting with one for the current global state and continuing according to each action's resolution. The number of global actions in an individual is determined by the depth set beforehand.

The bottom half of figure 7 shows a list of intended choices for each board per *round*. Each element $s_{r,p}$ refers to the branch number (global action alternative) to pick in global state $\langle r, \beta_p \rangle$, implying an intended global action $\alpha^{s_{r,p}}_{\beta_p}$. All intentions make up one growing (partial) global strategy s.

Candidate solutions are encoded real-valued: the orders in global actions are entirely and explicitly represented. This allows one mutation to change one order in an individual, which is very desirable since actions with similar orders are expected to have similar evaluations.

3.3.2 Initiation

The initial population consists of one individual, involving global actions to always hold all units. In fact, one global action is formed to hold all units in the current board configuration and that global action is the intention for all subsequent rounds, starting from the present.

Behold the following initiation example for the initial game state $\langle 0, \beta_0 \rangle$, playing 'Germany', and looking three rounds ahead. One global action $\alpha^0_{\beta_0}$ is created, involving *hold* orders for all units on board β_0:

$$\alpha^0_{\beta_0} = \lambda_{germany} \wedge \lambda_{germany}$$

with

$$\begin{aligned} \lambda_{germany} = \quad & issue(germany, hold(army, ber)) \\ \wedge \quad & issue(germany, hold(fleet, kie)) \\ \wedge \quad & issue(germany, hold(army, mun)) \end{aligned}$$

and $\lambda_{germany}$ a similar conjunction with *issue* predicates involving *hold* orders for all units belonging to empires other than 'Germany'.

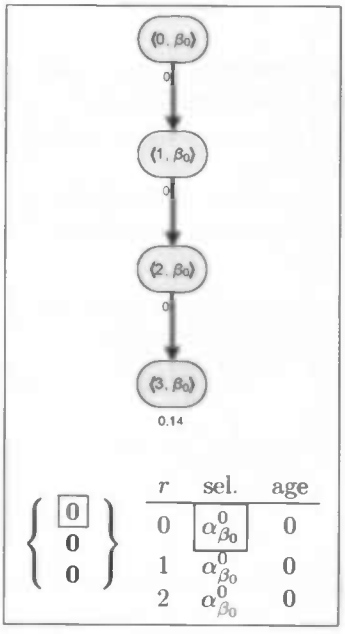

Figure 8: Representation of population initiation

The initial strategy is to choose $\alpha^0_{\beta_0}$, whenever board β_0 comes up, as depicted on the bottom left of figure 8: always choose branch 0. The search space in question is shown on the top, with all nodes involving board β_0 up to depth 3 and their global action branches. In the next sections we will get around the 0.14, the small table on the bottom right, and the cute little boxes.

3.3.3 Selection

The selection criterion on the individuals is based on the evaluation of the global states they bring about, represented by the leaf nodes in the currently known search space. The individual (series of global actions) that results in the best global state (leaf node) is sought by a depth-first-search. When enough intentions are known (set earlier on), strategy s determines *one* path from the current global state node to the best leaf node reachable, representing the global actions that compose the best individual in the pool.

The evaluation function at the leaf nodes is fairly simple. It counts the areas with production centers that Ares conquered so far. Ares' empire should either own the area or have a unit in it (in the latter case, Ares is very likely to own the area in the next season). Normalization over all conquered production centers (by anyone) yields the evaluations.

Just *one* individual is selected. A more complicated selection method seems unnecessary and it would probably just slow down the evolution process. This is merely conjecture, though.

The imminent evolution causes the components of the best individual to constantly change. Even more, changing just one intended global action might very well change subsequent global states, subsequent global actions, and thus the selected individual. Ares therefore keeps track of how many times each *global action* is evolved (its age), rather than counting evolution iterations.

Ares picks the youngest (least aged) global action in the selected individual to reproduce. Let's say that the rth global action in the selected individual (for round r) is the youngest and that the expected board configuration in that round is β_p. Since Ares intends to choose alternative $s_{r,p}$ at that point, it will use global action $\alpha^{s_{r,p}}_{\beta_p}$ for reproduction.

Continuing the example from section 3.3.2, the only available individual consists of three times global action $\alpha^0_{\beta_0}$. It results in global state $\langle 3, \beta_0 \rangle$, which is evaluated at, say, 0.14 (this is neutral, given there are seven players). We have no choice but to select this individual for the moment, as written under 'sel.' (selection) on the bottom right of figure 8. On the right side of each involved global action is its age. Obviously it is the same for each individual piece, since those pieces are the same. It should also be no surprise that age starts with 0. Ares picks the youngest global action in the selected individual, or the first if in doubt (for no particular reason). That is what the boxes in the strategy and the selected individual mark.

None of the individuals (or their global action parts) are discarded, mainly because their uselessness is hard to ground. A global action that cannot contribute to a good individual now just might do so later. The downside of keeping all global actions in memory is that it might slow down the selection algorithm dramatically when the pool gets bigger. This might be something to look into later.

3.3.4 Reproduction

As described in section 3.3.3 just one global action (limb) from just one individual is chosen for reproduction, possibly providing new limbs to create new individuals with.

Ares reproduces by means of mutation, for two main reasons. First, crossover (between two selected parents) is not likely to produce very good children. A good plan is more likely to improve by changing one or two steps, than to mix it with another. Plans are likely to be good because of the specific combination. Secondly, mutation is faster to calculate. With a task as complex as *evolving Diplomacy strategies* we might as well make the algorithm efficient wherever possible.

Only orders issued by Ares' empire are mutated (for now), hopefully towards better global actions for that empire. It produces a global action for global states with the same board configuration its parent had, say β_p. If it is among the $h+1$ existing global actions for that board $(\alpha^0_{\beta_p}, \ldots, \alpha^h_{\beta_p})$ the child is discarded. If not, it is named $\alpha^{h+1}_{\beta_p}$.

Consequences of global actions are resolved immediately. Whenever a new global action joins the limb pool, Ares asks Atlas to generate the resulting board. If this board is new (not currently stored), it is added to memory. Then it becomes an initial global action list with one global action to hold all units and an initial intention list to always choose that global action. In any case, the newly reproduced global action becomes a pointer to the board that results from it.

One important remark needs to be made here. Each global action has *one* pointer to the board its resolution brings about. In constructing individuals, these pointers will be used to determine in which board global actions result, regardless of the round. However, the resolution of a global action need not necessarily be exactly the same in each round; area ownership only changes at the end of Fall. We should have had *two* pointers, one to indicate the resulting board in *Spring* and one in *Fall*. This flaw is responsible for a slightly incorrect forecast on what intentions bring about. The brighter side is that Ares needs fewer resolution forecasts and that the evaluation function nullifies the flaw by regarding *owned* and *occupied* areas equally.

The addition of global action $\alpha^{h+1}_{\beta_p}$ represents a new branch from all nodes $\langle r, \beta_p \rangle$ (any r) in the search space. The grounds for intentions $s_{r,p}$ have thereby become incomplete. There might not be a direct need to reconsider them all, but if we *do* need one, we *should* reconsider it. For now, Ares simply clears intentions $s_{r,p}$ for all r and with p according to board β_p. This clearance makes s a partial strategy, even for the limited known part of the search space.

Due to the new options in the search space the selected individual might no longer be the best one constructible: it needs to be revised. Ares repeats the selection procedure from section 3.3.3 to establish the series of global actions that now leads to the best leaf node possible. How-

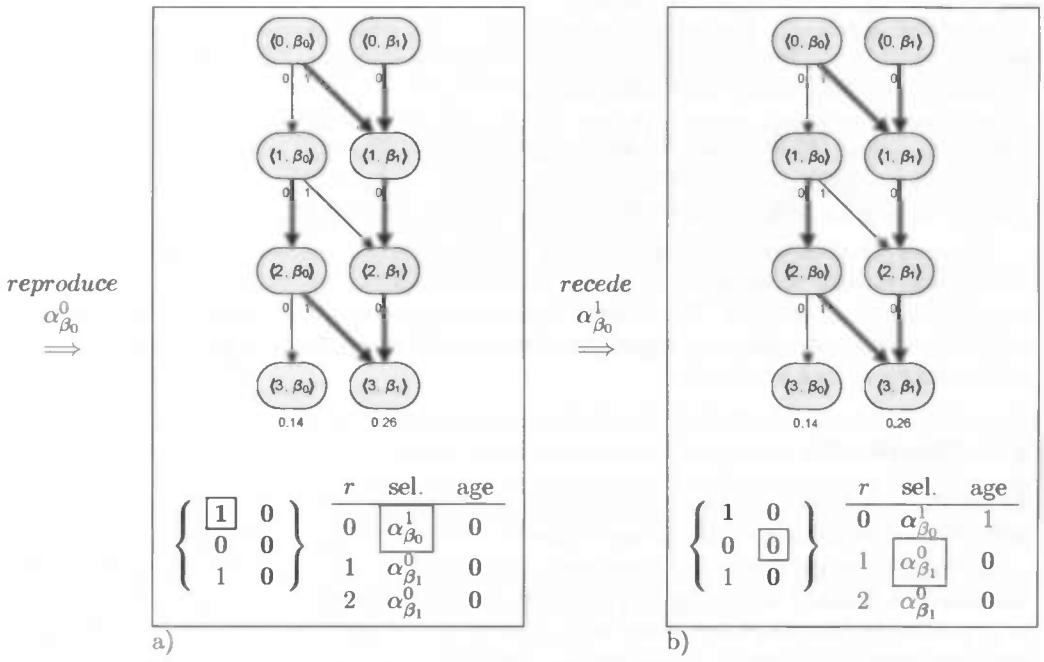

Figure 9: Example of the evolution process, first cycle
a) after reproduction and b) after recession

ever, branches from a node are only investigated when the intention for that node is unknown. Most of s was probably set before, explicitly stating the best path from that node on. The repetition of selection is thus not nearly as cumbersome as the title of its algorithm (*depth-first-search*) implies.

Mutating $\alpha^0_{\beta_0}$ from the previously proposed example could for instance produce the unique global action $\alpha^1_{\beta_0}$. The resulting board β_1 is yet unknown, thus added to memory with an initial *all-hold* global action $\alpha^0_{\beta_1}$ and an initial intention to use it whenever board β_1 comes up ($s_{0,1} = s_{1,1} = s_{2,1} = 0$). Since nodes $\langle r, \beta_0 \rangle$ (for all r) suddenly got a new branch, $s_{0,0}$ through $s_{2,0}$ are cleared, before re-selection starts.

The selection algorithm first needs to determine $s_{0,0}$ again. To do so, it needs to choose between α^0_{0,β_0} and α^1_{0,β_0}. The first yields the same choice in the next round, while the second eventually yields node $\langle 3, \beta_1 \rangle$, which is evaluated at, say, 0.26. The choice depends on $s_{1,0}$, which similarly depends on $s_{2,0}$. For the latter, Ares will choose branch 1, since it leads to a board with evaluation 0.26, while branch 0 would result in a board evaluation of 0.14. This makes the choice on $s_{1,0}$ indifferent: both branches lead to the better board. Ares does pick one, for no particular reason, say branch 0. Likewise the model now chooses $s_{0,0} = 1$.

Figure 9a shows the resulting strategy s (bottom left) and the search space belonging to it (at the top). The numbers in s refer to the thickened branches in the search space. The chosen individual is implied by strategy s: it leads from $\langle 0, \beta_0 \rangle$ via $\langle 1, \beta_1 \rangle$ and $\langle 2, \beta_1 \rangle$ to $\langle 3, \beta_1 \rangle$. The involved intentions in s are **bold** and the involved global actions are under 'sel.' on the bottom right.

New children do not necessarily lead to new global states. If we would continue the reproduction process for the next limb $\alpha^0_{\beta_1}$ in figure 9b, we might end up with a child $\alpha^1_{\beta_1}$ that leads

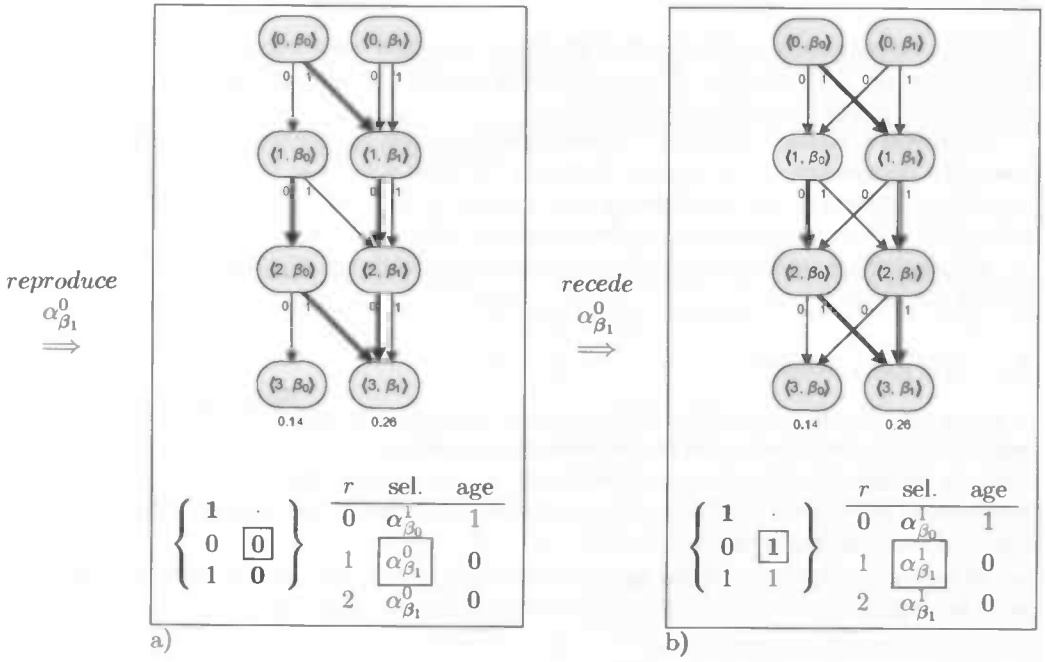

Figure 10: Example of the evolution process, second cycle
a) after reproduction and b) after recession

to the same next global state as its parent does, $\langle 2, \beta_1 \rangle$. Board β_1 is then not stored again, as figure 10a depicts.

3.3.5 Recession

Recession works in a similar way to reproduction in that it uses mutations. However, this time the orders issued by the six *opponents* are mutated, aiming at a better global action for them. It accounts for an appropriate counter λ to the involved orders for Ares' empire λ. The global action at hand does not produce *new* global actions like in reproduction: recession *changes* global actions.

The idea is to try to find the orders that best obstruct Ares' own intentions, assuming opponents intend to do so. This should worsen (recede) global actions as much as possible, accounting for plausible consequences to own intentions. Global actions should thus result in the worst evaluated board, given the own orders involved. If a particular iteration of *recession* did not actually lower the evaluation of the expected board configuration, the mutations are declined, leaving the global action unaltered. Recession *always* ages the global action at hand: an increment of one per iteration.

In figure 9a, the youngest limb of the best individual is $\alpha^1_{\beta_0}$. In the first example, recession might successfully change orders within this global action without changing its resolution. The composition of the best individual remains in this case. Receding $\alpha^1_{\beta_0}$ (from figure 9a) leaves the strategy intact (figure 9b), whatever recession took place.

Let us assume that the second cycle *does* change a global action such that its resolution changes. Recession on $\alpha^0_{\beta_1}$ from figure 10a, for instance, makes it point to the worse board β_0 (figure 10b). Intentions $s_{1,1}$ and $s_{2,1}$ are revised; both become 1.

3.3.6 Evolution

Together, reproduction and recession form the evolution mechanism that searches for the best local action for Ares' empire. The goodness of considered local actions λ is consolidated by their endurance of opposition attempts λ.

Evolution continues until the intended global actions for the expected global states have reached a certain age. At that point, choosing the local actions as they appear in the selected individual has made the inevitability of a desired global state plausible enough. The local action in the first global action in the best, mature individual is carried out.

All parameters on the number of generations, the number of mutations, and depth were set during simulation.

3.4 The game world

A game world was designed and implemented solely for the adjudicator model and the agent model to run in. It takes care of the communication between the agent and the adjudicator and with the database. It provides the structure of the game board and the script for the judge and the agent to perform in game simulations. In this work the game world was used to guide the simulations of numerous games.

In games against humans, the game world could be triggered by the interface when orders need to be resolved or artificially intelligent agents need to wake up.

3.5 Graphical user interface

We designed an internet game portal at our website *Diplomacy on-line* [6]. It was mainly used to see what was going on during the simulations, since raw data from the database was not all that intuitive.

The interface shows the orders given by each empire in each round and which of those orders were followed. Board resolutions are presented with different symbols to distinguish different order types. Clear arrows show the issued and followed moves and moving units actually move. Convoy paths are drawn through all areas they pass, based on the flags set by Atlas' board resolution stage (section 3.2.7). If necessary, the website triggers the C++ game world to let Ares think and Atlas adjudicate.

The website also allows human players to play against Ares or each other. Users may enter their orders in the games they participate in and even communicate with others, if the game variant allows that. Communication is not (yet) understood by Ares, though.

4 Simulation and results

As stated in section 1.3 the goal of this work is to make Atlas provide perfect board resolutions fast and to make Ares play fast and better than random. In other words, both models should be *accurate* and *efficient*. The next section describes the simulations to estimate these quality attributes and the results of those simulations.

4.1 Set up

The data for this work was gathered from three simulations of which a description is given below. The simulations were run on a 64 bit Dual Core Processor AMD server with 2 GB internal memory.

1. Adjudicator test cases

   Atlas was confronted with two sets of game situations to test its correctness. The first set consists of the 32 diagrams in the rulebook of diplomacy [8]. This makes up for a first check to see whether the adjudicator is able to process all order types. This set is used to explain Diplomacy resolution, but it does not cover all possible situations.

   The second set of game situations, gathered by Kruijswijk [17] is entirely devoted to testing automated adjudicators. It consists of 123 situations with any imaginable complexity. Furthermore, it deals with resolutions that seem obvious to humans, but require aspects of an adjudicator that are often overlooked. The set has been revised many times over the years, resulting in a prominent adjudicator test bench.

2. German openings

   We set up game openings with each empire in each game owning units at their starting positions [8], as in figure 1. Each game lasted for just one round and Ares played 'Germany' in each game. Ares repeatedly issued three orders for the three 'German' units at their starting positions 'Berlin', 'Kiel', and 'Munchen'.

   We used 13 different combinations of values for the parameters on solution age and search depth. The opening orders Ares played were logged to allow an opening frequency investigation. Additionally, Ares' average response time was determined per parameter setting.

   The opening frequencies were compared with the most common openings in 220 online Diplomacy games, collected by Theije [26]. Theije's analysis mostly agrees with a similar, earlier investigation by Agar on the most popular openings in 1913 games of Postal Diplomacy (via postal mail, letters) among human beings [1].

Table 1: Simulation iterations for German openings for each combination of parameters on solution age and search depth

	age 10	age 20	age 30	age 40
depth 1	50	50	50	50
depth 2	50	1	1	50
depth 3	50	1	1	
depth 4	1	1		

The number of iterations per parameter setting is shown in table 1. Seven runs involved 50 iterations of opening play, the other six entailed just one iteration. This is because the

response time for higher ages and depths grew extremely large (see section 4.2). Meanwhile, the enormous variations of the played openings in the completed 50-iteration runs did not suggest much importance to perform more of those (see section 4.3).

The significance of the response times in the single-iteration runs is much lower than those with 50 iterations. They do contribute to an image of response time dependencies and they help setting the parameters for longer simulations, like "Playing Diplomacy".

3. Playing Diplomacy

Ares was set up against an opponent model that randomly issues hold and move orders for each unit it controls with equal chance on each alternative. The opponent model never issues supports or convoys.

128 combinations exist to assign one of two models to each of seven players. To account for fair, significant results, 128 games, each with one of those combinations, were run. Equally represented, the two player models were on average 3.5 times present in a game, 448 instances per model in total.

Board resolutions that were made to *propel the game* were logged, including information on the performed stages and response times. Board resolutions that were made to *forecast order considerations of players* were *not* logged. Player statistics (like the course of each player's score) were derived from resolution logs.

4.2 Efficiency

Atlas

The test cases in simulation 1 were resolved in times varying from 0.1 to 10 milliseconds, with an average of approximately 1 millisecond. In general, however, these test cases contained fewer orders than games usually bring forth.

During simulation 3, the response times of 13323 board resolutions were logged, together with the last stage Atlas needed to decide on the success of all orders involved in each resolution (the conclusive stage). Appendix D shows the number of board resolutions that occurred per resolution time interval of 0.05 milliseconds per unit, per conclusive stage. Table 2 shows how the average resolution time per order was computed for the board resolutions with each conclusive stage.

Table 2: Computing the average resolution times per conclusive stage

conclusive stage	decision conditioning	predicate substitution	predicate supposition	any stage
(a) total number of boards	2543	10518	262	13323
(b) average resolution time (ms)	2.1	10.3	14.3	8.8
(c) average number of orders	32.8	32.1	32.2	32.3
(d) avg. res. time per order (ms)	0.06	0.32	0.44	0.27

Per conclusive stage, the resolution times were averaged over the number of boards with that conclusive stage (a). The resulting average resolution times (b) were divided by the average number of orders (c) for each conclusive stage. This should remove a possible relation between the number of orders and the conclusive stage and yields the average resolution times per order per conclusive stage (d).

Atlas performed 13323 board resolutions in an average of 8.8 ms per board. Most order decision sets (78.9 percent) were concluded after the predicate substitution stage. The simpler

cases were resolved much faster than the more complex cases. None of the resolutions required paradox elimination. It is assumed that convoy orders are required to create such complexity (e.g. test case 6.F.23 [17], resolved in appendix C). In this work, Ares was not able to issue convoy orders.

Ares

We estimated Ares' efficiency using simulation 2. The processing times for each used parameter setting is shown in table 3 and depicted in figure 11.

Table 3: Ares' German opening response times (in seconds) with various parameter settings

	age 10	age 20	age 30	age 40
depth 1	4.60	12.7	26.8	40.0
depth 2	54.1	225	503	3570
depth 3	482	15500	23700	
depth 4	6830	39700		

With 10 evolution cycles (for the final solution) and looking at direct consequences only (depth 1), Ares opens within 5 seconds. Higher age or depths show an enormous increase in processing time, reaching over 11 hours.

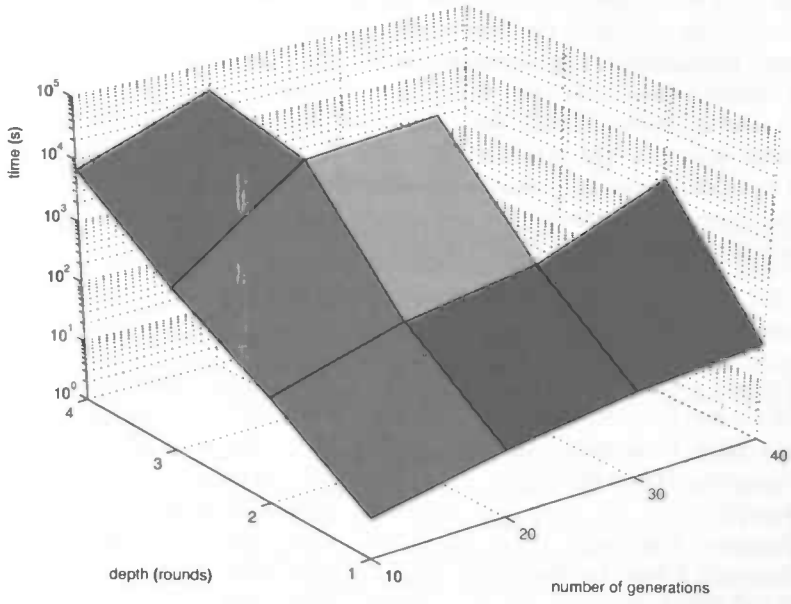

Figure 11: Ares' response time, opening for Germany related to age and depth

The approximate straightness of the plane in the logarithmic axes system of figure 11 suggests an exponential relation between age and processing time and between depth and processing time. Considering the significance of the six single-iteration runs, combined with suspected variance in the available server processor power we cannot conclude anything at this point. However, these findings do suggest to choose age and depth as low as possible in the 128 games of simulation 3; just to have the results within this life time.

4.3 Accuracy

Atlas

In simulation 1 Atlas resolved all 155 test cases according to the specifications of this work. Verifications were based on the game rules [8] (129 cases), clarifications from the Diplomacy Players Technical Guide (DPTG) [5] (three cases), Szykman's paradox elimination criteria [25] (eleven cases), Kruijswijk's preference [17] (two cases) and the order clarity demands in this work (section 1.2.4) (ten cases). Appendix E lists exactly which test case is correct according to which author.

Ten cases deviate from what Kruijswijk calls "2000 rulebook / DATC compliant" [17]. According to this standard, move orders with incomplete coast specifications are accepted in some cases (test cases 6.B.2, 6.B.7, 6.B.8, 6.B.10, and 6.B.12). Also, impossible move orders are always *illegal*, thus ignored, replaced by hold orders and capable of receiving hold support (test cases 6.D.28 through 6.D.32). This work *always* demands issues of *complete* orders and only regards move orders as *illegal* when the target area does not exist. Hence, these cases only comply with the specifications in this work. Since Ares is not even capable of issuing illegal orders of any kind, the discussion on the correct resolution of illegal orders is not that important.

Ares

Ares used 127 different openings during the seven 50-iteration runs of simulation 2 (single-iteration runs were solely used for response time measurements). Appendix F lists the occurrence of each opening with each used parameter setting. Table 4 shows how many times Ares played the most popular openings according to literature [1, 26].

The area abbreviations under *ber*, *kie*, and *mun* refer to the target area of an issued *move* order for the unit in 'Berlin', 'Kiel', and 'Munchen' respectively. An 'H' under *ber*, *kie*, and *mun* denotes the issue of a *hold* order for the unit in 'Berlin', 'Kiel', and 'Munchen' respectively.

Ares did not issue popular openings significantly more often than less popular openings. Vice versa, the openings Ares favors (by just around 2 percent) are mostly not among the highly popular openings in literature, as table 5 shows.

Table 4: Most popular German openings in literature

Name	*ber*	*kie*	*mun*	literature	Ares
Blitzkrieg, Danish Variation	*kie*	*den*	*ruh*	36.8%	1.7%
Anschluß, Burgundy Push	*kie*	*den*	*bur*	23.2%	0.6%
Blitzkrieg, Dutch Variation	*kie*	*hol*	*ruh*	12.7%	1.4%
Anschluß	*kie*	*den*	H	7.7%	1.4%
Burgundy Opening	*kie*	*hol*	*bur*	4.6%	1.4%
Bismarck, Dutch Grab	*kie*	*hol*	*tyr*	2.3%	0.6%
Polish Blitzkrieg	*sil*	*den*	*ruh*	1.8%	0.9%
Anschluß, Dutch Variation	*kie*	*hol*	H	1.4%	0.0%
Bismarck, Danish Grab	*kie*	*den*	*tyr*	1.4%	0.9%
Blitzkrieg, Baltic Variation	*kie*	*bal*	*ruh*	1.4%	0.0%
Barbarossa	*pru*	*den*	*sil*	1.4%	2.3%
Blitzkrieg, Heligoland Variation	*kie*	*hel*	*ruh*	0.9%	0.0%
Anschluß, Ruhr Variation	*mun*	*den*	*ruh*	0.9%	0.9%
Anschluß, Silesian Variation	*kie*	*den*	*sil*	0.9%	0.6%
(other)				2.6%	87.4%

Table 5: Most popular German openings by Ares

Name	*ber*	*kie*	*mun*	literature	Ares
Barbarossa	*pru*	*den*	*sil*	1.4%	2.3%
(no name)	H	*den*	*bur*	0.0%	2.0%
Berlin defense, Ruhr Variation	H	*den*	*ruh*	0.0%	1.7%
(no name)	*kie*	*den*	*kie*	0.0%	1.7%
Blitzkrieg, Danish Variation	*kie*	*den*	*ruh*	36.8%	1.7%
Dutch opening, Silesia variation	*kie*	*hol*	*sil*	0.0%	1.7%
(no name)	*mun*	*hol*	*sil*	0.0%	1.7%
(no name)	*mun*	*hol*	*tyr*	0.0%	1.7%
(no name)	*pru*	*hol*	*tyr*	0.0%	1.7%
(no name)	*sil*	*den*	*kie*	0.0%	1.7%
(other)				61.8%	82.0%

There are two ways to lose a game. A player might get *eliminated* by losing all of his units or he might *survive* the game ordeals until someone else claims victory. Although there is neither an award for *elimination* nor for *survival*, the latter is preferred.

The only way to *win* a game is to conquer more than half of the production centers. When none of the remaining players succeeds in reaching this goal, they may agree upon a *draw*, in which they share victory.

The results of the 896 agents in the 128 games from simulation 3 are listed in appendix G. The accomplishments of the two agent models are summarized in table 6.

Table 6: Results on Ares and the random agent in 128 games

agent	win	draw	survival	elimination	score
Ares	105	75	226	42	123.0
random	4	4	119	321	5.0

All Ares agents and random playing agents were assigned scores as described in section 1.2.6 (1 point for a victory and 1 point split equally among participants of a draw). Ares won 105 games (points) and lost 4; the other 19 games were pre-ended in round 218, resulting in a draw. Since, in the 19 draws, random players were eliminated much more often than Ares players (17 draws were between Ares agents only), Ares claimed most of those points too. Note that both agents received one victory point for free, since they both participated in one game opposing themselves only.

The progress of each agent's victories was indicated by calculating his score for each round, assuming all games would end right then. The sums of both agents' scores were normalized, yielding the fractions of all games each agent had won so far. Since victory is usually proclaimed after a second round (a whole fictional year) scores were determined once a year. Ares' participation in the overall victory in years 1901 through 2009 (rounds 0 through 216) is in table 7. A more graphical representation can be found in figure 12. The graph was cut off at year 2000, since no score changes were reported after that.

Table 7: Normalized scores for Ares per game year. The left column refers to the last digit of each year in the decennia in the top row.

	1900	1910	1920	1930	1940	1950	1960	1970	1980	1990	2000
0		.545	.684	.809	.872	.911	.944	.947	.953	.956	.961
1	.500	.556	.698	.815	.880	.915	.944	.948	.953	.956	.961
2	.500	.569	.719	.822	.888	.923	.944	.948	.953	.958	.961
3	.500	.580	.744	.823	.893	.926	.944	.948	.953	.958	.961
4	.506	.589	.752	.829	.899	.932	.944	.948	.955	.961	.961
5	.508	.593	.764	.835	.899	.936	.946	.950	.957	.961	.961
6	.514	.611	.775	.846	.906	.943	.946	.948	.956	.961	.961
7	.523	.626	.779	.848	.907	.945	.946	.953	.956	.961	.961
8	.531	.642	.791	.853	.908	.945	.946	.953	.956	.961	.961
9	.537	.671	.797	.863	.909	.945	.946	.953	.956	.961	.961

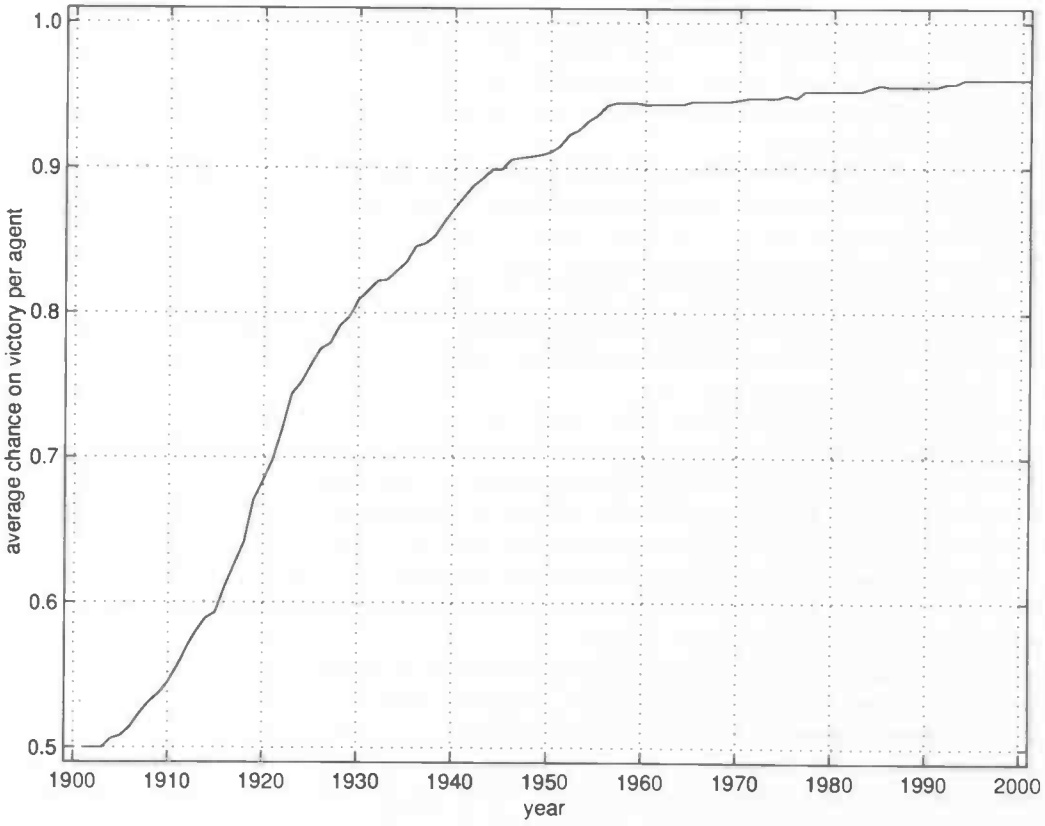

Figure 12: Normalized scores for Ares per game year

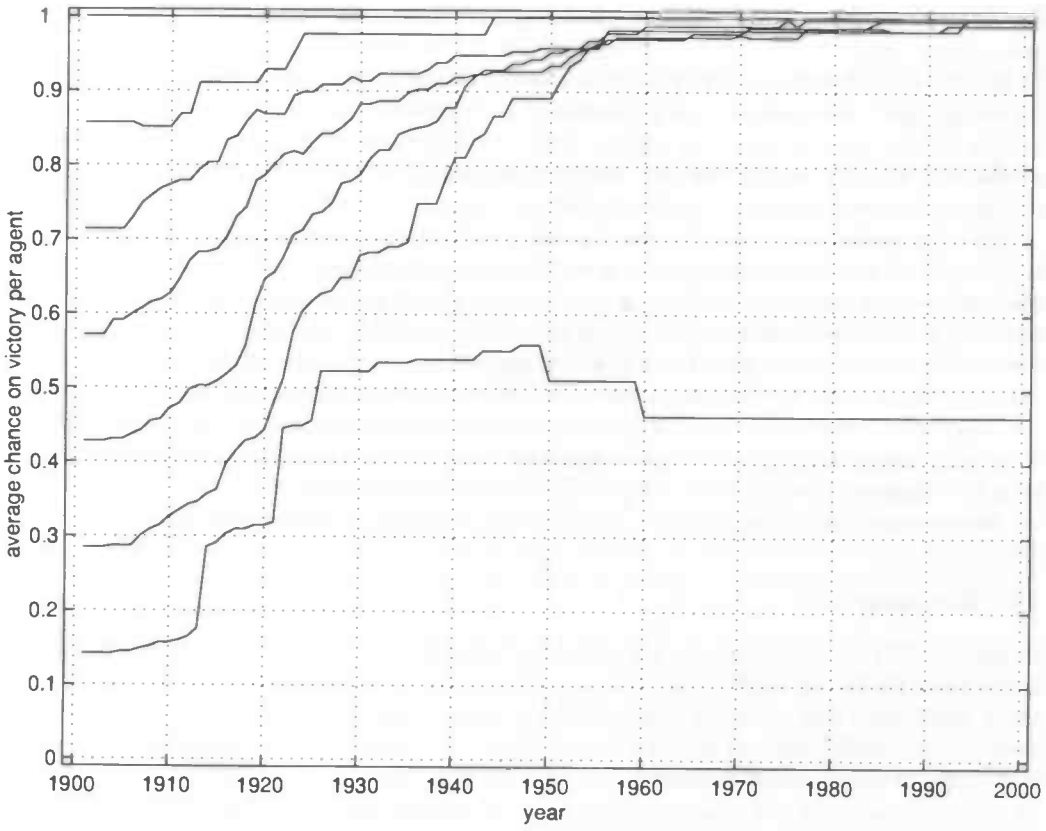

Figure 13: Normalized scores for one through seven Ares agents per game year

The bottom line in figure 13 shows Ares' progress against six random players. On average, one Ares wins almost half of the games against six random players. The second line from the bottom shows the collective progress of two Ares agents against five random players, indicating an almost inevitable victory for one of the Ares agents. More instances of Ares in a game leave random players even less chance.

The average progress per empire, regardless of the agent model that is playing it, is shown in appendix H.

5 Discussion, conclusions and future work

In the previous chapters, we showed how the resolution of Diplomacy orders can be modeled by using logic. We designed an algorithm that determines the conditions to the success of issued orders. Several stages of logical derivation eliminate conditions until a conclusion is reached on the success of all orders. Simple cases are quickly deduced to a solution; the more complex cases endure more complex deduction stages.

We also modeled strategy forming by using evolutionary computing. Our algorithm represents global action alternatives for several global states it expects to meet. The algorithm distinguishes between its own local action in each global action and those of others. Evaluations of global states that series of global actions (to a certain depth) bring forth, provide the genetic fitness of those series. The algorithm repeatedly breeds new global action alternatives from the fittest series by mutating (and improving) its own local actions. Meanwhile, a similar process mutates the local actions of opponents, yielding a recession of the fittest global actions. The resolution of each global action is assumed to be a plausible consequence to the involved own local action, allowing a fair comparison of global action alternatives.

The next sections entail a contemplation of the obtained results from chapter 4.

5.1 Retrospection

In section 1.2.4 we explained the contradiction that adjudication according to the Avalon Hill game rules [8] brings forth. Contradictory cases arise in which we are torn between two or more resolutions that all either meet or violate these rules. In the latter case we must choose between successful *supports* and successful *moves*. We believe that in actual combat an army movement is not expected to reach its destination if the army is obstructed before it gets there. More likely, supports will spasmodically fixate all current positions. Therefore, to solve many paradoxical cases, we proposed the *support sustain* resolution rule that meets our intuition on a natural course of war.

The *support sustain* rule removes the contradiction from most situations that would have been contradictory otherwise. Also, it allows for a very basic elimination of remaining paradoxes, failing all involved moves. Szykman [25] once proposed a rule that suggests to change paradoxical convoy orders to hold orders. Although this rule imposes the same resolution on paradoxical situations, it is much harder to incorporate in an algorithm than our solution. It requires either an explicit, beforehand paradox detection mechanism, or specific paradox adjudication when simpler adjudication fails.

Kruijswijk [17] describes two approaches to automated adjudication. The first is *sequence-based* in which order aspects are tackled in a pre-defined sequence. Each order success decision is made in the last step, when no more uncertainties exist to prevent it. The second approach is *decision-based* and continuously tries to make decisions that are free from impediments.

This work introduced the *condition-based* and the *logic-based* adjudication algorithm as alternatives to Kruijswijk's two approaches. The first determines decision dependencies as soon as they emerge. Recursive condition processing ultimately leads to the *known* facts that order successes rely upon. Logic-based adjudication builds on logical representations of the conditions to the success of each order. Logic reasoning mechanisms sooner or later yield the solution to all decisions; simpler cases sooner, more complex cases later.

Our logic-based adjudicator model 'Atlas' (section 3.2) provides correct resolutions to all test cases the Diplomacy community has so far thought of (section 4.3). The average response time of our adjudicator model is insignificant compared to that of our agent model Ares (section 4.2). So far, other artificially intelligent Diplomacy playing agents made *no* use of resolution forecasts (section 1.5), while Ares does.

Atlas' resolution times are related to the complexity of the cases at hand (section 4.2). We believe that the logic-based structure of our model is partly responsible for this. Simple cases are resolved quickly because complex operations are not applied. Unfortunately, existing adjudication models came without efficiency indications [5, 17], so an absolute comparison with those could not be made.

Human adjudication is probably best mimicked by a *condition-based* algorithm that tries to make the decision on the success of an order by investigating its dependencies. For simpler cases humans might incline towards a *decision-based* approach and make decisions when intuition says so. The more complex cases are left for puzzling later on, much like *logic-based* algorithms. Usually humans do not seize explicit logic, since the solution is apparent. Our algorithm always uses a logic-based approach, making it not very humanoid. This was not the aim of this work and we now conclude it never should be. Although humans may think their case-specific approach is very smart, the seamlessly self-evident pre-selection could become harshly cumbersome when we put it in a model.

So far, significant Diplomacy player models have been *agent-based* (e.g. The Israeli Diplomat [16]), *tree-based* (e.g. The Bordeaux Diplomat [19]), *evaluation-based* (e.g. DumbBot [22, 24], DiploBot [24], and Project20M [14, 24]), or *goal-based* (e.g. HaAI [11, 24]). Literature presented Project20M and HaAI as the most prominent agent models at the moment. We demonstrated a (new) way of approaching the enormous interaction problem of Diplomacy in an *evolution-based* way with 'Ares' (section 3.3).

A relation between Ares' first orders, playing 'Germany', and the same orders by humans could not be proven. The frequency of Ares' preferred openings did not significantly stand out.

Looking ahead through great depths does not seem very lucrative. We were unable to improve the consistency of Ares' orders by increasing its search depth. Possibly this is due to the explosive increase in uncertainty when assuming more opposing actions. The imperfectness of Ares' information might build upon itself.

Although a comparison with any competent agent model could not be made, Ares plays Diplomacy better than a random playing agent (section 4.3). Its actions are therefore sensible and, to some degree, artificially intelligent.

In our game simulations Ares produced strategies with depth two, of the tenth generation. Its response times were around one minute per action, much less than the time limit on order writing for human players (section 4.2). Human players are normally obliged to write their order within five minutes time. In battles against humans Ares' efficiency should suffice, but for a computational agent it is quite low.

We believe that human's way of playing Diplomacy lies somewhere between *evaluation-based*, *goal-based*, and *evolution-based*. On the basis of the most interesting areas (*evaluations*) players come up with areas they would like to conquer (*goals*). They reconcile goals with each other, yielding several strategies they might pursue. Players may repeatedly consider minor strategy adjustments by trying to predict the consequence of each alternative (*evolution*). And without communication possibilities they need some assumption on the plans of other players. One might build an opponent model (see [9]) or simply assume the most unfortunate opposition.

In any unpredictable world artificial intelligence *should* reach for human experience to remedy its lack of insight. Huff et al gave us *evaluation*, Haard brought *goals*, and this work may have grabbed the third piece of the AI Diplomacy puzzle: *evolution*. Maybe future work could create a hybrid trinity.

5.2 Conclusions

From our results from chapter 4 and the retrospection of section 5.1 we draw the following conclusions:

- Atlas is ultimately accurate, given the restrictions on order notation.
- Atlas is very efficient, allowing a firm base of many resolutions to artificially intelligent Diplomacy playing.
- Logic-based resolution seems to be a good alternative to sequence-based and decision-based resolution.
- Ares plays games of Diplomacy better than a random playing model.
- Ares generates orders faster than humans normally do.
- Evolutionary computing is a hopeful technique in automated strategy forming.

Looking back at the aim of this work (section 1.3) we may conclude that Atlas and Ares comply with our specifications. Our goals on the quality of these models have thereby been achieved.

5.3 Application

Most surprisingly, many internet Diplomacy games make use of human game masters to adjudicate orders. In our opinion, automated adjudication would do a much quicker and more reliable job, given the right algorithm.

We have high hopes for the application of logically deductive algorithms, similar to the one used for Atlas, on planning problems like the making of school time tables. Given a number of teachers, class rooms, students, and subjects, we want to put together consistent time tables. It seems that the problem is usually too big to incorporate subordinate, but desirable, aspects like fair deals of free afternoons. If we could formalize all these aspects, logical deduction might lead to a time table that *does* suit all demands or at least reduce the problem to one in which we can easily choose one that does.

Similarly, logical deduction could smoothly and efficiently take care of unforeseen airport schedule changes. A plane's departure delay is often caused by the inability to agree an earlier departure with the existing, tight schedule. Even if an earlier departure is easily feasible with minor schedule adjustments, the search space is too large to find the solution fast enough. With formalizations of the demands and the decisions that make up the problem, logical deduction might be the answer to logistics, policy-making, and corporate management.

Since Ares showed less sensible intentions for the more distant future, strategy evolution algorithms might attain better results on strategic games (like Stratego, Scotland Yard) than they did on Diplomacy. Stratego is played with just two players and Scotland Yard has much less possibilities per turn than Diplomacy has. Both games thus have a much smaller search space. Strategy forming with great depths is therefore much more feasible in these games than it is in Diplomacy, for human players and for artificially intelligent models.

5.4 Further research

In this work we focused on binary decisions: an order either *succeeds* or *fails*. The fields of logistics and management cope with many possibilities on *who* does *what* at *which time*: solutions to problems in these fields require /it multi-decisive logical deduction models. Continuous variables (like time) might be incorporated by defining discrete time-slots.

Many aspects of our strategy evolution algorithm may be optimized. Limits on the number of stored board situations or on the number of action alternatives per board situation might improve Ares' efficiency. Basically, the used algorithm was too inefficient to allow many parameter adjustment iterations.

Our strategy evolution algorithm may improve by better combining the available knowledge. The algorithm could initiate intentions in new situations with intentions in situations it already knows. Even more, the intentions for similar board situations could produce a more abstract intention, literally bringing strategy forming to a higher level.

Normally, Diplomacy includes negotiation and is therefore mainly based on trust. We could investigate the influence of trust on a proper Diplomacy agent algorithm. Extracting knowledge from negotiations and utilizing it might be an interesting field as well. Another step could be a research on the impact of negotiation itself. A player might act differently, just because he *knows* he can negotiate. The logical internal representation of our algorithm would perfectly allow for such extensions.

It would be very interesting to investigate the gained performance by combining techniques that have proven to work separately. We could combine strategy evolution techniques with area evaluation processing like in Project20M [14, 24] and/or with goal generation like in HaAI [11, 24]. The best artificially intelligent Diplomacy player might be a hybrid solution that generates *goals* that aim at the best *evaluations* and that *evolves* strategies to fit those goals.

References

[1] Agar, S. The most popular UK postal Dipomacy openings. http://www.diplomacy-archive.com/resources/postal/openings.htm.

[2] Barwise, J.; Etchemendy, J. *The Language of First-Order Logic*. Center for the Study of Language and Information, Stanford (CA), third edition, 1992.

[3] Benthem, J.F.A.K. van; Ditmarsch, H.P. van; Ketting J.; Lodder L.S.; Meyer-Viol W.P.M. *Logica voor informatici*. Open University, Heerlen, second edition, 1994.

[4] Binmore, K. *Fun and Games*. D.C. Heath and Company, Lexington (MA), 1992.

[5] Black, K.; et al. The Diplomacy player's technical guide. http://www.diplomacy.org.il/dptg.txt, 1998.

[6] Booijink, B.A.M. Diplomacy on-line. http://www.diplomacy.cc, 2004-2005.

[7] Brandenburger, A.; Nalebuff, B. *Coopetition*. Doubleday, New York (NY), 1996.

[8] Calhamar, A.B. *The Rules of Diplomacy*. The Avalon Hill Game Company, Pawtucket (RI), 4th edition, 2000.

[9] Donkers, H.H.L.M. *Nosce Hostem; Searching with Opponent Models*. PhD thesis, IKAT, Maastricht, 2003.

[10] Douma, F.E. Kwartetten: Logica en strategie voor het kaartspel. Master's thesis, Artificial Intelligence, University of Groningen, Groningen, 2002.

[11] Håård, F.; Johansson, S.J. Multi-agent tactics in Diplomacy. *AAMAS 2005*, 2005.

[12] Hoek, W. van der; Meyer, J.-J.Ch. *Epistemic Logic for AI and Computer Science*. Cambridge Tracts in Theoretical Computer Science No 41, Cambridge University Press, Cambridge (MA), 1995.

[13] Holland, J.H. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor (MI), 1975.

[14] Huff, A.; Chan, V.; Tondelier L.; Bundred D.; Egan C. Automated negotiation in the game of Diplomacy. http://www.doc.ic.ac.uk/~ajh02/diplomacy, 2005.

[15] jDip Web Page. http://jdip.sourceforge.net, 2004.

[16] Kraus, S.; Lehmann, D. Designing and building a negotiating automated agent. *Computational Intelligence*, 11:132–171, 1995.

[17] Kruijswijk, L.B. Diplomacy adjudicator test cases. http://web.inter.nl.net/users/L.B.Kruijswijk, 2001-2004.

[18] Loeb, D.E. Challenges in multi-player gaming by computers. http://www.diplom.org/Zine/S1995M/Loeb/Project.html, 1995.

[19] Loeb, D.E. Stable winning coalitions. In Nowakowski, N.J., editor, *Games of no chance: combinatorial games at MSRI*, volume 29 of *Mathematical Sciences Research Institute publications*, pages 451–471. Cambridge University Press, 1996.

[20] Microprose. The hasbro Diplomacy game. http://www.microprose.com, 2000.

[21] Mitchell, M. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge (MA), 1996.

[22] Norman, D. David's Diplomacy AI page. http://www.ellought.demon.co.uk/dipai, 2003.

[23] Roberts, B. Man'chi Diplomacy AI. http://ca.geocities.com/bmr335/manchi.html, 2004.

[24] Rose, A. Diplomacy AI centre. http://www.starblood.org/daide, 2003-2004.

[25] Szykman, S. Eliminating the paradox in Diplomacy. http://www.diplom.org/Zine/F1999R/Debate/paradox.html, 1999.

[26] Theije, C. de. Analysis of openings and their results. http://home.planet.nl/~ciesdeth/home/home.html, 2005.

A PostgreSQL data structure

User
name
address
postal code
city
country
email
avatar
birthday
sex
username
password
registered
code
status
logins
lastlogin
lastactivity
game
empire
won
lost
total
points
id

Area
code
name
type
unit_x
unit_y
flood_fills
army_adjacent
fleet_adjacent
army_adj_nums
fleet_adj_nums
connections
city
id

Game
name
austria_type
germany_type
turkey_type
italia_type
france_type
england_type
russia_type
austria_user
germany_user
turkey_user
italia_user
france_user
england_user
russia_user
phase
diplomatic_time
retreat_time
gain_lose_time
deadline
type
status
progress
id

Board
name
phase
occupied_by
units
resolve_time
resolve_stage
id

Order
game
empire
phase
from_num
to_num
via_num
action
status
id

B Condition-based adjudication example: Diagram 29.

Units on the board

- England: army London; fleet England; fleet North Sea
- France: fleet Brest; fleet Irish Sea

Formal representation of orders

- England:
  $issue(england, move(army, lon, bel))$
  $issue(england, convoy(fleet.\ eng, move(army, lon, bel)))$
  $issue(england, convoy(fleet.\ nth, move(army, lon, bel)))$
- France:
  $issue(france, move(army, bre, eng))$
  $issue(france, support(fleet, iri, move(army, bre, eng)))$

Figure 14: Diagram 29.

This diagram shows that an army convoyed using alternate convoy orders reaches its destination as long as at least one convoy route remains open [8]. Figure 14 shows the board configuration.

For each predicate the algorithm uses a specific function to determine the conditions the decision (satisfaction) of that predicate depends on. Most of these predicates have similar meanings to the algorithm as they have to the logic based algorithm (see section 2.1). Additionally, $under\_attack(u, a)$ denotes if a *move* has been *issued* to area a, $dislodge(u, a)$ denotes if unit u at area a is unable to *hold* and $path(a, b, c)$ denotes if a convoy path exists from area a to area b, via area c.

The algorithm stores determined (truth) values. If more conditions turn out to depend on them, the algorithm retrieves them from memory instead of processing them again. The exact implementation is beyond the scope of this thesis; as described in section 3.2, the condition based algorithm was discarded for its limited capabilities.

Table 8: Symbol explanation in the condition based adjudication

$[a]$	the algorithm (re)starts with the order for the unit in area a
$\varphi\,?$	predicate φ is processed
$\varphi = V(\varphi)$	the algorithm satisfies predicate φ with truth value $V(\varphi)$
$\varphi \doteq V(\varphi)$	truth value $V(\varphi)$ for predicate φ is re-used
$\varphi\ \langle a \rangle$	an order for the unit in area a is ignored in determining $V(\varphi)$
$f\,?$	function f is processed
$f = n$	the algorithm satisfies function f with value n
$f \doteq n$	value n for function f is re-used

Below is a log dump of the condition-based adjudication algorithm resolving the orders above. The used symbols are explained in table 8.

$[eng]$

$follow(convoy(fleet, eng, move(army, lon, bel)))$?

$attainable(move(fleet, bre, eng))$?

$adjacent(fleet, bre, eng)$?

$adjacent(fleet, bre, eng) = true$

$attainable(move(fleet, bre, eng)) = true$

$[strength(move(fleet, bre, eng)) > resistance(hold(fleet, eng))]$?

$strength(move(fleet, bre, eng))$?

$under\_attack(fleet, iri)\ \langle eng \rangle$?

$under\_attack(fleet, iri)\ \langle eng \rangle = false$

$dislodge(fleet, iri)$?

$dislodge(fleet, iri) = false$

$strength(move(fleet, bre, eng)) = 2$

$resistance(hold(fleet, eng))$?

$resistance(hold(fleet, eng)) = 1$

$[strength(move(fleet, bre, eng)) > resistance(hold(fleet, eng))] = true$

$follow(move(fleet, bre, eng))$?

$attainable(move(fleet, bre, eng))$?

$attainable(move(fleet, bre, eng)) \doteq true$

$[strength(move(fleet, bre, eng)) > resistance(hold(fleet, eng))]$?

$[strength(move(fleet, bre, eng)) > resistance(hold(fleet, eng))] \doteq true$

$follow(move(fleet, bre, eng)) = true$

$follow(convoy(fleet, eng, move(army, lon, bel))) = false$

$[nth]$

$follow(convoy(fleet, nth, move(army, lon, bel)))$?

$follow(convoy(fleet, nth, move(army, lon, bel))) = true$

$[lon]$

$follow(move(army, lon, bel))$?

$attainable(move(army, lon, bel))$?

$adjacent(army, lon, bel)$?

$adjacent(army, lon, bel) = false$

$path(lon, bel, lon)$?

$path(lon, bel, eng)$?

$dislodge(fleet, eng)$?

$dislodge(fleet, eng) \doteq true$

$path(lon, bel, eng) = false$

$path(lon, bel, nth)$?

$dislodge(fleet, nth)$?

$dislodge(fleet, nth) \doteq false$

$path(lon, bel, nth) = true$

$path(lon, bel, lon) = true$

$attainable(move(army, lon, bel)) = true$

$follow(move(army, lon, bel)) = true$

$[iri]$

$follow(support(fleet, iri, move(fleet, bre, eng)))$?

$under\_attack(fleet, iri)\ \langle eng \rangle$?

$under\_attack(fleet, iri)\ \langle eng \rangle \doteq false$

$dislodge(fleet, iri)$?

$dislodge(fleet, iri) \doteq false$

$follow(support(fleet, iri, move(fleet, bre, eng))) = true$

C Logic-based adjudication example: Test case 6.F.23

Test case 6.F.23 is one of Kruijswijk's most complex cases, involving a second order paradox with two exclusive convoys [17]. The board configuration and the issued orders are written below.

Units on the board

- England: fleet Edinburgh; fleet Yorkshire
- France: army Brest; fleet English Channel
- Germany: fleet Belgium; fleet London
- Italy: fleet Mid-Atlantic Ocean; fleet Irish Sea
- Russia: army Norway; fleet North Sea

Formal representation of orders

- England:
  $move(fleet, edi, nth)$
  $support(fleet, yor, move(fleet, edi, nth))$
- France:
  $move(army, bre, lon)$
  $convoy(fleet, eng, move(army, bre, lon))$
- Germany:
  $support(fleet, bel, hold(fleet, eng))$
  $support(fleet, lon, hold(fleet, nth))$
- Italy:
  $move(fleet, mid, eng)$
  $support(fleet, iri, move(fleet, mid, eng))$
- Russia:
  $move(army, nwy, bel)$
  $convoy(fleet, nth, move(army, nwy, bel))$

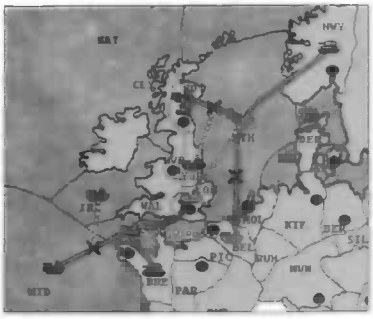

Figure 15: Test case 6.F.23

Applying the game rules only (without any paradox rule), there are two consistent resolutions, but where the two convoys do not fail or succeed at the same time. In one resolution, the convoy in the English Channel is dislodged by the fleet in the Mid-Atlantic Ocean, while the convoy in the North Sea succeeds. In the other resolution, it is the other way around. The convoy in the North Sea is dislodged by the fleet in Edinburgh, while the convoy in the English Channel succeeds.

Applying Szykman's rule, the convoying armies fail to move and the supports are not cut. Because of the failure to cut the support, no fleet succeeds to move. We will see that the additional rules proposed in this work (section 1.2.4) ensure the same resolution.

The condition rules described in paragraph 3.2.2 imply the following success sentences for the above orders:

1. $follow(convoy(fleet, eng, move(army, bre, lon))) \longleftrightarrow$
   $\neg follow(move(fleet, mid, eng))$

2. $follow(convoy(fleet, nth, move(army, nwy, bel))) \longleftrightarrow$
   $\neg follow(move(fleet, edi, nth))$

3. $follow(move(army, bre, lon)) \longleftrightarrow$
   $attainable(move(army, bre, lon)) \wedge$
   $[strength(move(army, bre, lon)) > strength(hold(fleet, lon))]$

4. $follow(move(army, nwy, bel)) \longleftrightarrow$
   $attainable(move(army, nwy, bel)) \wedge$
   $[strength(move(army, nwy, bel)) > strength(hold(fleet, bel))]$

5. $follow(move(fleet, edi, nth)) \longleftrightarrow$
   $attainable(move(fleet, edi, nth)) \wedge$
   $[strength(move(fleet, edi, nth)) > strength(hold(fleet, nth))]$

6. $follow(move(fleet, mid, eng)) \longleftrightarrow$
   $attainable(move(fleet, mid, eng)) \wedge$
   $[strength(move(fleet, mid, eng)) > strength(hold(fleet, eng))]$

7. $follow(support(fleet, bel, hold(fleet, eng))) \longleftrightarrow$
   $\neg attainable(move(army, nwy, bel))$

8. $follow(support(fleet, iri, move(fleet, mid, eng))) \longleftrightarrow true$

9. $follow(support(fleet, lon, hold(fleet, nth))) \longleftrightarrow$
   $\neg attainable(move(army, bre, lon))$

10. $follow(support(fleet, yor, move(fleet, edi, nth))) \longleftrightarrow true$

Since Atlas replaces occurrences of *attainable* right away and distributes support conditions over strength variables, the above set of sentences is never actually generated. It is presented here to shed some light on how Atlas gets to the sentences below. What actually results from the condition determination stage is written below. Notice that functions are no longer of any importance now that their appropriate sentence sums represent them.

1. $follow(convoy(fleet, eng, move(army, bre, lon))) \longleftrightarrow \neg follow(move(fleet, mid, eng))$
2. $follow(convoy(fleet, nth, move(army, nwy, bel))) \longleftrightarrow \neg follow(move(fleet, edi, nth))$
3. $follow(move(army, bre, lon)) \longleftrightarrow follow(convoy(fleet, eng, move(army, bre, lon)))\wedge [\Sigma\emptyset > \Sigma\emptyset]$
4. $follow(move(army, nwy, bel)) \longleftrightarrow follow(convoy(fleet, nth, move(army, nwy, bel)))\wedge [\Sigma\emptyset > \Sigma\emptyset]$
5. $follow(move(fleet, edi, nth)) \longleftrightarrow true\wedge [\Sigma true > \Sigma\neg follow(convoy(fleet, eng, move(army, bre, lon)))]$
6. $follow(move(fleet, mid, eng)) \longleftrightarrow true\wedge [\Sigma true > \Sigma\neg follow(convoy(fleet, nth, move(army, nwy, bel)))]$
7. $follow(support(fleet, bel, hold(fleet, eng))) \longleftrightarrow \neg follow(convoy(fleet, nth, move(army, nwy, bel)))$
8. $follow(support(fleet, iri, move(fleet, mid, eng))) \longleftrightarrow true$
9. $follow(support(fleet, lon, hold(fleet, nth))) \longleftrightarrow \neg follow(convoy(fleet, eng, move(army, bre, lon)))$
10. $follow(support(fleet, yor, move(fleet, edi, nth))) \longleftrightarrow true$

1. $eng \leftrightarrow \neg mid$
2. $nth \leftrightarrow \neg edi$
3. $bre \leftrightarrow eng\wedge [\Sigma\emptyset > \Sigma\emptyset]$
4. $nwy \leftrightarrow nth\wedge [\Sigma\emptyset > \Sigma\emptyset]$
5. $edi \leftrightarrow true\wedge [\Sigma true > \Sigma\neg eng]$
6. $mid \leftrightarrow true\wedge [\Sigma true > \Sigma\neg nth]$
7. $bel \leftrightarrow \neg nth$
8. $iri \leftrightarrow true$
9. $lon \leftrightarrow \neg eng$
10. $yor \leftrightarrow true$

On the right is an abbreviation of the sentences on the left to make life a little easier. Each area code resembles the $follow$ predicates of the order that has been committed to the unit in that area.

The first deductions remove unnecessary predicates and variables.

1. $follow(convoy(fleet, eng, move(army, bre, lon))) \longleftrightarrow \neg follow(move(fleet, mid, eng))$
2. $follow(convoy(fleet, nth, move(army, nwy, bel))) \longleftrightarrow \neg follow(move(fleet, edi, nth))$
3. $follow(move(army, bre, lon)) \longleftrightarrow false$
4. $follow(move(army, nwy, bel)) \longleftrightarrow false$
5. $follow(move(fleet, edi, nth)) \longleftrightarrow follow(convoy(fleet, eng, move(army, bre, lon)))$
6. $follow(move(fleet, mid, eng)) \longleftrightarrow follow(convoy(fleet, nth, move(army, nwy, bel)))$
7. $follow(support(fleet, bel, hold(fleet, eng))) \longleftrightarrow \neg follow(convoy(fleet, nth, move(army, nwy, bel)))$
8. $follow(support(fleet, iri, move(fleet, mid, eng))) \longleftrightarrow true$
9. $follow(support(fleet, lon, hold(fleet, nth))) \longleftrightarrow \neg follow(convoy(fleet, eng, move(army, bre, lon)))$
10. $follow(support(fleet, yor, move(fleet, edi, nth))) \longleftrightarrow true$

1. $eng \leftrightarrow \neg mid$
2. $nth \leftrightarrow \neg edi$
3. $bre \leftrightarrow false$
4. $nwy \leftrightarrow false$
5. $edi \leftrightarrow eng$
6. $mid \leftrightarrow nth$
7. $bel \leftrightarrow \neg nth$
8. $iri \leftrightarrow true$
9. $lon \leftrightarrow \neg eng$
10. $yor \leftrightarrow true$

After substitution of predicates (and more deductions where possible):

1. $follow(convoy(fleet, eng, move(army, bre, lon))) \longleftrightarrow \neg follow(move(fleet, mid, eng))$
2. $follow(convoy(fleet, nth, move(army, nwy, bel))) \longleftrightarrow follow(move(fleet, mid, eng))$
3. $follow(move(army, bre, lon)) \longleftrightarrow false$
4. $follow(move(army, nwy, bel)) \longleftrightarrow false$
5. $follow(move(fleet, edi, nth)) \longleftrightarrow \neg follow(move(fleet, mid, eng))$
6. $follow(move(fleet, mid, eng)) \longleftrightarrow follow(move(fleet, mid, eng))$
7. $follow(support(fleet, bel, hold(fleet, eng))) \longleftrightarrow \neg follow(move(fleet, mid, eng))$
8. $follow(support(fleet, iri, move(fleet, mid, eng))) \longleftrightarrow true$
9. $follow(support(fleet, lon, hold(fleet, nth))) \longleftrightarrow follow(move(fleet, mid, eng))$
10. $follow(support(fleet, yor, move(fleet, edi, nth))) \longleftrightarrow true$

1. $eng \leftrightarrow \neg mid$
2. $nth \leftrightarrow mid$
3. $bre \leftrightarrow false$
4. $nwy \leftrightarrow false$
5. $edi \leftrightarrow \neg mid$
6. $mid \leftrightarrow mid$
7. $bel \leftrightarrow \neg mid$
8. $iri \leftrightarrow true$
9. $lon \leftrightarrow mid$
10. $yor \leftrightarrow true$

Apparently, one or more order successes depend on themselves. Atlas assumes the predicate $follow(move(fleet, mid, eng))$ to be $true$, resulting in (deducted):

1. $follow(convoy(fleet, eng, move(army, bre, lon))) \longleftrightarrow false$
2. $follow(convoy(fleet, nth, move(army, nwy, bel))) \longleftrightarrow true$
3. $follow(move(army, bre, lon)) \longleftrightarrow false$
4. $follow(move(army, nwy, bel)) \longleftrightarrow false$
5. $follow(move(fleet, edi, nth)) \longleftrightarrow false$
6. $follow(move(fleet, mid, eng)) \longleftrightarrow true$
7. $follow(support(fleet, bel, hold(fleet, eng))) \longleftrightarrow false$
8. $follow(support(fleet, iri, move(fleet, mid, eng))) \longleftrightarrow true$
9. $follow(support(fleet, lon, hold(fleet, nth))) \longleftrightarrow true$
10. $follow(support(fleet, yor, move(fleet, edi, nth))) \longleftrightarrow true$

1. $eng \leftrightarrow false$
2. $nth \leftrightarrow true$
3. $bre \leftrightarrow false$
4. $nwy \leftrightarrow false$
5. $edi \leftrightarrow false$
6. $mid \leftrightarrow true$
7. $bel \leftrightarrow false$
8. $iri \leftrightarrow true$
9. $lon \leftrightarrow true$
10. $yor \leftrightarrow true$

Assuming $follow(move(fleet, mid, eng))$ to be $false$ results in (deducted):

1. $follow(convoy(fleet, eng, move(army, bre, lon))) \longleftrightarrow true$
2. $follow(convoy(fleet, nth, move(army, nwy, bel))) \longleftrightarrow false$
3. $follow(move(army, bre, lon)) \longleftrightarrow false$
4. $follow(move(army, nwy, bel)) \longleftrightarrow false$
5. $follow(move(fleet, edi, nth)) \longleftrightarrow true$
6. $follow(move(fleet, mid, eng)) \longleftrightarrow false$
7. $follow(support(fleet, bel, hold(fleet, eng))) \longleftrightarrow true$
8. $follow(support(fleet, iri, move(fleet, mid, eng))) \longleftrightarrow true$
9. $follow(support(fleet, lon, hold(fleet, nth))) \longleftrightarrow false$
10. $follow(support(fleet, yor, move(fleet, edi, nth))) \longleftrightarrow true$

1. $eng \leftrightarrow true$
2. $nth \leftrightarrow false$
3. $bre \leftrightarrow false$
4. $nwy \leftrightarrow false$
5. $edi \leftrightarrow true$
6. $mid \leftrightarrow false$
7. $bel \leftrightarrow true$
8. $iri \leftrightarrow true$
9. $lon \leftrightarrow false$
10. $yor \leftrightarrow true$

No assumption for $follow(move(fleet, mid, eng))$ results in the success of all dependable supports (orders 7 and 9), thus none is chosen.

Any depending *move* fails (orders 5 and 6), any depending *convoy* or *support* succeeds (orders 1, 2, 7, and 9). The truth value of the other four orders (3, 4, 8, and 10) was already determined.

1. $follow(convoy(fleet, eng, move(army, bre, lon))) \longleftrightarrow true$
2. $follow(convoy(fleet, nth, move(army, nwy, bel))) \longleftrightarrow true$
3. $follow(move(army, bre, lon)) \longleftrightarrow false$
4. $follow(move(army, nwy, bel)) \longleftrightarrow false$
5. $follow(move(fleet, edi, nth)) \longleftrightarrow false$
6. $follow(move(fleet, mid, eng)) \longleftrightarrow false$
7. $follow(support(fleet, bel, hold(fleet, eng))) \longleftrightarrow true$
8. $follow(support(fleet, iri, move(fleet, mid, eng))) \longleftrightarrow true$
9. $follow(support(fleet, lon, hold(fleet, nth))) \longleftrightarrow true$
10. $follow(support(fleet, yor, move(fleet, edi, nth))) \longleftrightarrow true$

1. $eng \leftrightarrow true$
2. $nth \leftrightarrow true$
3. $bre \leftrightarrow false$
4. $nwy \leftrightarrow false$
5. $edi \leftrightarrow false$
6. $mid \leftrightarrow false$
7. $bel \leftrightarrow true$
8. $iri \leftrightarrow true$
9. $lon \leftrightarrow true$
10. $yor \leftrightarrow true$

The application of these order successes to the board is relatively easy, to say the least. No unit moves, so all units are simply mapped to the same areas on the new board. Area ownership changes are not tested in this run, neither are retreats.

D Resolution statistics

Table 9: Number of board resolutions per conclusive stage, per time interval

resolution time per unit (ms)	decision conditioning	predicate substitution	predicate supposition
0.00-0.05	1355	0	0
0.05-0.10	685	185	0
0.10-0.15	266	649	0
0.15-0.20	126	1024	0
0.20-0.25	59	1341	12
0.25-0.30	29	1568	20
0.30-0.35	14	1586	21
0.35-0.40	4	1581	46
0.40-0.45	2	1122	35
0.45-0.50	2	705	50
0.50-0.55	0	401	34
0.55-0.60	0	192	22
0.60-0.65	1	90	12
0.65-0.70	0	41	4
0.70-0.75	0	23	4
0.75-0.80	0	3	2
0.80-0.85	0	3	0
0.85-0.90	0	1	0
0.90-0.95	0	2	0
0.95-1.00	0	0	0
1.00-1.05	0	0	0
1.05-1.10	0	0	0
1.10-1.15	0	1	0
total	2543	10518	262

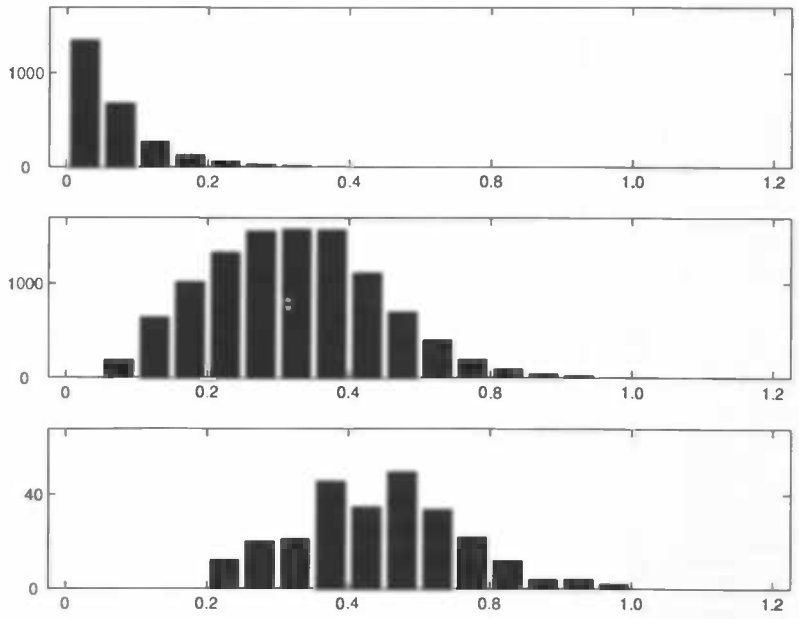

Figure 16: Resolutions per conclusive stage and time interval

E Resolution test cases

Table 10: Response times and verifications of test case resolutions;

the fifth column states the author whose preference was met for each case.

case	title	time (ns)	solution stage	author
	rulebook order adjudication			
1	army movement	368	decision conditioning	Calhamar, 2000
2	fleet movement from sea	454	decision conditioning	Calhamar, 2000
3	fleet movement from coast	507	decision conditioning	Calhamar, 2000
4	moves of equal strength to the same area fail	924	decision conditioning	Calhamar, 2000
5	one unit not moving can stop other units from moving	1136	predicate substitution	Calhamar, 2000
6	units cannot trade places without the use of convoy	838	decision conditioning	Calhamar, 2000
7	three or more units can rotate areas provided none trade places	1650	predicate supposition	Calhamar, 2000
8	simple support by an army	862	decision conditioning	Calhamar, 2000
9	simple support by a fleet	843	decision conditioning	Calhamar, 2000
10	move support in standoffs	1635	decision conditioning	Calhamar, 2000
11	hold support in standoffs	1073	decision conditioning	Calhamar, 2000
12	a dislodged unit can still cause a standoff	2290	decision conditioning	Calhamar, 2000
13	a dislodged unit has no effect on the area that dislodged it (1)	3429	predicate substitution	Calhamar, 2000
14	a dislodged unit has no effect on the area that dislodged it (2)	5122	predicate substitution	Calhamar, 2000
15	support is cut by an attack from an area different from the one where support is given	1346	decision conditioning	Calhamar, 2000
16	support is cut is the unit giving support is dislodged (1)	1543	predicate substitution	Calhamar, 2000
17	support is cut is the unit giving support is dislodged (2)	2926	predicate substitution	Calhamar, 2000
18	a unit being dislodged by one area can still cut support in another	1956	decision conditioning	Calhamar, 2000
19	convoy order	912	predicate substitution	Calhamar, 2000
20	convoying across several sea areas	1640	predicate substitution	Calhamar, 2000
21	dislodgement of a fleet in a convoy causes the convoy to fail	2106	predicate substitution	Calhamar, 2000
22	an empire cannot dislodge own units or support that (1)	777	decision conditioning	Calhamar, 2000
23	an empire cannot dislodge own units or support that (2)	2696	predicate substitution	Calhamar, 2000
24	an empire cannot dislodge own units or support that (3)	1378	decision conditioning	Calhamar, 2000
25	an empire cannot dislodge own units or support that (4)	2842	predicate substitution	Calhamar, 2000
26	an empire cannot dislodge own units or support that (5)	1838	decision conditioning	Calhamar, 2000
27	self standoff	763	decision conditioning	Calhamar, 2000
28	exchanging places via convoy	1163	predicate supposition	Calhamar, 2000
29	more than one convoy route	1099	predicate substitution	Calhamar, 2000
30	a convoyed army does not cut the support of an attack on one of the fleets in its convoy	1287	predicate substitution	Calhamar, 2000
31	a successfully convoyed army does cut the support of an attack on an alternate convoy route (1)	1780	predicate substitution	Calhamar, 2000
32	a successfully convoyed army does cut the support of an attack on an alternate convoy route (2)	3405	predicate substitution	Calhamar, 2000
	basic checks			
6.A.1	moving to an area that is not a neighbor	190	decision conditioning	Calhamar, 2000
6.A.2	move army to sea	186	decision conditioning	Calhamar, 2000
6.A.3	move fleet to land	189	decision conditioning	Calhamar, 2000
6.A.4	move to own sector	327	decision conditioning	Calhamar, 2000
6.A.5	move to own sector with convoy	1134	predicate substitution	Calhamar, 2000
6.A.6	ordering a unit of another country	319	decision conditioning	Calhamar, 2000
6.A.7	only armies can be convoyed	250	decision conditioning	Calhamar, 2000
6.A.8	support to hold yourself is not possible	759	predicate substitution	Calhamar, 2000
6.A.9	fleets must follow coast if not on sea	190	decision conditioning	Calhamar, 2000
6.A.10	support on unreachable destination not possible	623	decision conditioning	Calhamar, 2000
6.A.11	simple bounce	513	decision conditioning	Calhamar, 2000
6.A.12	bounce of three units	1030	decision conditioning	Calhamar, 2000
	coastal issues			
6.B.1	moving with unspecified coast when coast is necessary	192	decision conditioning	Calhamar, 2000
6.B.2	moving with unspecified coast when coast is not necessary	189	decision conditioning	Boonjink, 2005
6.B.3	moving with wrong coast when coast is not necessary	191	decision conditioning	Kruijswijk, 2004
6.B.4	support to unreachable coast allowed	780	decision conditioning	Calhamar, 2000
6.B.5	support from unreachable coast not allowed	620	decision conditioning	Calhamar, 2000
6.B.6	support can be cut with other coast	883	decision conditioning	Calhamar, 2000
6.B.7	supporting with unspecified coast	1117	decision conditioning	Boonjink, 2005
6.B.8	supporting with unspecified coast when only one coast is possible	1118	decision conditioning	Boonjink, 2005
6.B.9	supporting with wrong coast	934	decision conditioning	Kruijswijk, 2004
6.B.10	unit ordered with wrong coast	326	decision conditioning	Boonjink, 2005
6.B.11	coast cannot be ordered to change	320	decision conditioning	Calhamar, 2000
6.B.12	army movement with coastal specification	236	decision conditioning	Boonjink, 2005
6.B.13	coastal crawl not allowed	530	decision conditioning	Calhamar, 2000
	circular movement			
6.C.1	three army circular movement	661	predicate supposition	Calhamar, 2000
6.C.2	three army circular movement with support	894	predicate supposition	Calhamar, 2000
6.C.3	a disrupted three army circular movement	889	predicate substitution	Calhamar, 2000
6.C.4	a circular movement with attacked convoy	2361	predicate supposition	Calhamar, 2000
6.C.5	a disrupted circular movement due to dislodged convoy	2488	predicate substitution	Calhamar, 2000
6.C.6	two armies with two convoys	1161	predicate supposition	Calhamar, 2000
6.C.7	disrupted unit swap	1899	predicate substitution	Calhamar, 2000

case	title	time (ns)	solution stage	author
	supports and dislodges			
6.D.1	supported hold can prevent dislodgement	602	decision conditioning	Calhamar, 2000
6.D.2	a move cuts support on hold	876	decision conditioning	Calhamar, 2000
6.D.3	a move cuts support on move	766	decision conditioning	Calhamar, 2000
6.D.4	support to hold on unit supporting a hold allowed	696	decision conditioning	Calhamar, 2000
6.D.5	support to hold on unit supporting a move allowed	913	decision conditioning	Calhamar, 2000
6.D.6	support to hold on convoying unit allowed	1131	predicate substitution	Calhamar, 2000
6.D.7	support to hold on moving unit not allowed	1195	decision conditioning	Calhamar, 2000
6.D.8	failed convoy cannot receive hold support	778	decision conditioning	Calhamar, 2000
6.D.9	support to move on holding unit not allowed	539	decision conditioning	Calhamar, 2000
6.D.10	self dislodgment prohibited	333	decision conditioning	Calhamar, 2000
6.D.11	no self dislodgment of returning unit	985	predicate substitution	Calhamar, 2000
6.D.12	supporting a foreign unit to dislodge own unit prohibited	435	decision conditioning	Calhamar, 2000
6.D.13	supporting a foreign unit to dislodge a returning own unit prohibited	1089	predicate substitution	Calhamar, 2000
6.D.14	supporting a foreign unit is not enough to prevent dislodgement	769	decision conditioning	Calhamar, 2000
6.D.15	defender cannot cut support for attack on itself	858	predicate substitution	Calhamar, 2000
6.D.16	convoying a unit dislodging a unit of same power is allowed	728	predicate substitution	Calhamar, 2000
6.D.17	dislodgement cuts supports	1984	predicate substitution	Calhamar, 2000
6.D.18	a surviving unit will sustain support	2093	predicate substitution	Calhamar, 2000
6.D.19	even when surviving is in alternative way	1098	predicate substitution	Calhamar, 2000
6.D.20	unit cannot cut support of its own country	582	decision conditioning	Calhamar, 2000
6.D.21	dislodging does not cancel a support cut	1345	decision conditioning	Calhamar, 2000
6.D.22	impossible fleet move cannot be supported	802	decision conditioning	Calhamar, 2000
6.D.23	impossible coast move cannot be supported	678	decision conditioning	Calhamar, 2000
6.D.24	impossible army move cannot be supported	636	decision conditioning	Calhamar, 2000
6.D.25	failing hold support can be supported	719	decision conditioning	Calhamar, 2000
6.D.26	failing move support can be supported	749	decision conditioning	Calhamar, 2000
6.D.27	failing convoy can be supported	892	predicate substitution	Calhamar, 2000
6.D.28	impossible move and support	696	decision conditioning	Booijink, 2005
6.D.29	move to impossible coast and support	735	decision conditioning	Booijink, 2005
6.D.30	move without coast and support	658	decision conditioning	Booijink, 2005
6.D.31	a tricky impossible support	351	decision conditioning	Booijink, 2005
6.D.32	a missing fleet	736	decision conditioning	Booijink, 2005
6.D.33	unwanted support allowed	1222	predicate substitution	Calhamar, 2000
6.D.34	support targeting own area not allowed	2434	predicate substitution	Calhamar, 2000
	head to head battles and beleaguered garrison			
6.E.1	dislodged unit has no effect on attacker's area	2296	predicate substitution	Calhamar, 2000
6.E.2	no self dislodgement in head to head battle	676	decision conditioning	Calhamar, 2000
6.E.3	no help in dislodging own unit	1391	predicate substitution	Calhamar, 2000
6.E.4	non-dislodged loser has still effect	7242	predicate substitution	Calhamar, 2000
6.E.5	loser dislodged by another army has still effect	8297	predicate substitution	Calhamar, 2000
6.E.6	not dislodge because of own support has still effect	4006	predicate substitution	Calhamar, 2000
6.E.7	no self dislodgement with beleaguered garrison	1738	decision conditioning	Calhamar, 2000
6.E.8	no self dislodgement with beleaguered garrison and head to head battle	3681	predicate substitution	Calhamar, 2000
6.E.9	almost self dislodgement with beleaguered garrison	2180	predicate substitution	Calhamar, 2000
6.E.10	almost circular movement with no self dislodgement with beleaguered garrison	2700	predicate substitution	Calhamar, 2000
6.E.11	no self dislodgement with beleaguered garrison, unit swap with adjacent convoying and two coasts	4649	predicate supposition	Calhamar, 2000
6.E.12	support on attack on own unit can be used for other means	2298	predicate substitution	Calhamar, 2000
6.E.13	three way beleaguered garrison	3005	decision conditioning	Calhamar, 2000
6.E.14	illegal head to head battle can still defend	462	predicate substitution	Calhamar, 2000
6.E.15	the friendly head to head battle	7219	predicate substitution	Calhamar, 2000
	convoys			
6.F.1	no convoy in coastal areas	1042	decision conditioning	Calhamar, 2000
6.F.2	an army being convoyed can bounce as normal	882	predicate substitution	Calhamar, 2000
6.F.3	an army being convoyed can receive support	1370	predicate substitution	Calhamar, 2000
6.F.4	an attacked convoy is not disrupted	624	predicate substitution	Calhamar, 2000
6.F.5	a beleaguered convoy is not disrupted	2352	predicate substitution	Calhamar, 2000
6.F.6	dislodged convoy does not cut support	2935	predicate substitution	Calhamar, 2000
6.F.7	dislodged convoy does not cause contested area	969	predicate substitution	Calhamar, 2000
6.F.8	dislodged convoy does not cause a bounce	1567	predicate substitution	Calhamar, 2000
6.F.9	dislodge of multi-route convoy	1177	predicate substitution	Calhamar, 2000
6.F.10	dislodge of multi-route convoy with foreign fleet	1220	predicate substitution	Calhamar, 2000
6.F.11	dislodge of multi-route convoy with only foreign fleets	1203	predicate substitution	Calhamar, 2000
6.F.12	dislodged convoying fleet not on route	1163	predicate substitution	Calhamar, 2000
6.F.13	the unwanted alternative	1232	predicate substitution	Calhamar, 2000
6.F.14	simple convoy paradox	1333	predicate substitution	Szykman, 1999
6.F.15	simple convoy paradox with additional convoy	2449	predicate substitution	Szykman, 1999
6.F.16	Pandin's paradox	2972	predicate substitution	Calhamar, 2000
6.F.17	Pandin's extended paradox	4656	predicate supposition	Szykman, 1999
6.F.18	betrayal paradox	2676	predicate supposition	Szykman, 1999
6.F.19	multi-route convoy disruption paradox	2033	predicate substitution	Szykman, 1999
6.F.20	unwanted multi-route convoy paradox	2575	predicate substitution	Szykman, 1999
6.F.21	dad's army convoy	2948	predicate substitution	Szykman, 1999
6.F.22	second order paradox with two resolutions	7274	predicate supposition	Szykman, 1999
6.F.23	second order paradox with two exclusive convoys	22816	paradox elimination	Szykman, 1999
6.F.24	second order paradox with no resolution	6202	predicate supposition	Szykman, 1999
	convoying to adjacent places			
6.G.1	two units can swap places by convoy	1420	predicate supposition	Calhamar, 2000
6.G.2	kidnapping an army	1399	predicate supposition	Calhamar, 2000
6.G.3	kidnapping with a disrupted convoy	923	predicate substitution	Calhamar, 2000
6.G.4	kidnapping with a disrupted convoy and opposite move	2361	predicate substitution	Calhamar, 2000
6.G.5	swapping with intent	2325	predicate supposition	Calhamar, 2000
6.G.6	swapping with unintended intent	2787	predicate supposition	Calhamar, 2000
6.G.7	swapping with illegal intent	1895	predicate supposition	Calhamar, 2000
6.G.8	explicit convoy that is not there	639	predicate substitution	Calhamar, 2000
6.G.9	swapped or dislodged?	1628	predicate substitution	Calhamar, 2000
6.G.10	swapped or a head to head battle?	4760	predicate substitution	DPTG, 1998
6.G.11	a convoy to an adjacent place with a paradox	1459	predicate substitution	Szykman, 1999
6.G.12	swapping two units with two convoys	4171	predicate supposition	Calhamar, 2000
6.G.13	support cut on attack on itself via convoy	1194	predicate substitution	Calhamar, 2000
6.G.14	bounce by convoy to adjacent place	4848	predicate substitution	DPTG, 1998
6.G.15	bounce and dislodge with double convoy	2627	predicate substitution	DPTG, 1998
6.G.16	the two unit in one area bug, moving by convoy	3595	predicate substitution	Calhamar, 2000
6.G.17	the two unit in one area bug, moving over land	3843	predicate substitution	Calhamar, 2000
6.G.18	the two unit in one area bug, with double convoy	5056	predicate substitution	Calhamar, 2000

F Opening frequencies for Germany

Table 11: Occurrences of each opening play with each used parameter setting; The first number in the pairs on top states the solution age, the second is search depth.

ber	*kie*	*mun*	10, 1	10, 2	10, 3	20, 1	30, 1	40, 1	40, 2	total
H	*ber*	*ruh*				1				1
H	*den*	*ber*		2		1				3
H	*den*	*boh*			1	1		1		3
H	*den*	*bur*	2		2			2	1	7
H	*den*	*kie*				1	1			2
H	*den*	*ruh*	1	2			2	1		6
H	*den*	*sil*				1			2	3
H	*den*	*tyr*			1		1		1	3
H	*den*	S *ber* H		3					1	4
H	*hol*	*ber*			1					1
H	*hol*	*boh*					1		1	2
H	*hol*	*bur*	1		2			1	1	5
H	*hol*	*tyr*			1	1			1	3
kie	H	*kie*					1			1
kie	*bal*	*boh*				1				1
kie	*ber*	*kie*			1					1
kie	*den*	H	1			1	2		1	5
kie	*den*	*ber*		1			1	1		3
kie	*den*	*boh*	1	1	2			1		5
kie	*den*	*bur*		2						2
kie	*den*	*kie*			1	1	2	1	1	6
kie	*den*	*ruh*	2	1	1	1			1	6
kie	*den*	*sil*	1				1			2
kie	*den*	*tyr*	1			2				3
kie	*den*	S *ber* H			1					1
kie	*hel*	S *ber kie*						1		1
kie	*hol*	*ber*				2			1	3
kie	*hol*	*boh*				1	2			3
kie	*hol*	*bur*	1	1	1			1	1	5
kie	*hol*	*kie*	1							1
kie	*hol*	*ruh*						5		5
kie	*hol*	*sil*		1	1	1		2	1	6
kie	*hol*	*tyr*				1	1			2
kie	*hol*	S *ber* H		1					1	2
kie	*hol*	S *ber kie*				1		1	1	3
mun	*bal*	*ber*			1					1
mun	*ber*	*bur*					1			1
mun	*ber*	S *kie ber*			1					1
mun	*den*	H	1			1				2
mun	*den*	*ber*	1	1		1				3
mun	*den*	*boh*	1			1				2
mun	*den*	*bur*			1			1	2	4
mun	*den*	*kie*	1	2						3
mun	*den*	*ruh*			2	1				3
mun	*den*	*sil*	2			1		2		5
mun	*den*	*tyr*				1		1		2
mun	*den*	S *ber* H							1	1
mun	*den*	S *ber kie*						1		1
mun	*hol*	H	2						2	4
mun	*hol*	*ber*		1	1				1	3
mun	*hol*	*boh*	3	1		1				5
mun	*hol*	*bur*							1	1
mun	*hol*	*kie*		2			1		1	4
mun	*hol*	*ruh*	1		1		1	2		5
mun	*hol*	*sil*	3				2		1	6
mun	*hol*	*tyr*			1	1	2		2	6
mun	*hol*	S *kie* H	1							1
mun	*hol*	S *ber sil*		1				1		2
mun	*hol*	S *kie ber*				1				1

ber	kie	mun	10, 1	10, 2	10, 3	20, 1	30, 1	40, 1	40, 2	total
pru	H	tyr		1						1
pru	den	H				1	2	1	1	5
pru	den	ber		3	2					5
pru	den	boh		3						3
pru	den	bur	1			1			2	4
pru	den	kie	1	2				1		4
pru	den	ruh	1		1		1	1	1	5
pru	den	sil	2	2	1			2	1	8
pru	den	tyr				1		1	2	4
pru	hol	H			1		1	1	1	4
pru	hol	ber			1			1		2
pru	hol	boh			2				1	3
pru	hol	bur	1			1				2
pru	hol	kie	1	1		1			1	4
pru	hol	ruh	1		1		1		1	4
pru	hol	sil		1	1		2			4
pru	hol	tyr	2				1	2	1	6
pru	hol	S ber H				1				1
pru	hol	S ber sil						1		1
pru	hol	S kie ber				1				1
sil	H	ruh			1				1	2
sil	bal	tyr			1					1
sil	den	H	1							1
sil	den	ber		1		2	1	1		5
sil	den	boh		1		1	1	1		4
sil	den	bur		1	1		2		1	5
sil	den	kie	1		1		1	3		6
sil	den	ruh	1		2					3
sil	den	sil		2			1	1		4
sil	den	tyr	1		1		2			4
sil	den	S ber sil		1			1		1	3
sil	hel	S kie H		1						1
sil	hol	H		1						1
sil	hol	ber		2					1	3
sil	hol	boh	1			1	1			3
sil	hol	bur	1		1	1			2	5
sil	hol	kie		1	1					2
sil	hol	ruh			1		2	1		4
sil	hol	sil	1		1	1	2			5
sil	hol	tyr	1		1		1			3
sil	hol	S ber H				1				1
sil	hol	S ber sil				1			1	2
S mun H	den	H			1	1	1		1	4
S mun H	den	bur						1		1
S mun H	den	kie				1				1
S mun H	den	sil	1					1	1	3
S mun H	den	S ber kie	1							1
S mun H	den	S ber sil							1	1
S mun H	hol	H					1			1
S mun H	hol	ber					1			1
S mun H	hol	boh				1				1
S mun H	hol	bur						1		1
S mun H	hol	S ber sil						2		2
S kie H	den	kie			1					1
S kie H	den	ruh				1				1
S kie H	den	S ber H		1						1
S kie H	hol	bur			1					1
S kie H	S ber H	boh	1							1
S munkie	ber	kie						1		1
S munkie	den	kie				1				1
S munkie	hol	S ber H					1			1
S munsil	den	bur		1						1
S munsil	den	ruh				1				1
S munsil	den	sil	1	1		1				3
S munsil	den	S kie ber							1	1
S munsil	hol	ber				1				1
S munsil	hol	tyr					1			1
S munsil	hol	S ber H			1					1

G Game simulation endings

Table 12: Game simulation endings

game	Austria	Germany	Turkey	Italy	France	England	Russia	status	round
1.	rand. *surv.*	rand. *elim.*	rand. *surv.*	rand. *elim.*	rand. *win*	rand. *elim.*	rand. *elim.*	finished	162
2.	rand. *elim.*	rand. *elim.*	rand. *elim.*	rand. *surv.*	rand. *surv.*	rand. *surv.*	Ares *win*	finished	26
3.	rand. *elim.*	rand. *elim.*	rand. *surv.*	rand. *elim.*	rand. *win*	Ares *surv.*	rand. *elim.*	finished	98
4.	rand. *elim.*	rand. *elim.*	rand. *surv.*	rand. *surv.*	rand. *surv.*	Ares *surv.*	Ares *win*	finished	30
5.	rand. *elim.*	rand. *elim.*	rand. *elim.*	rand. *elim.*	Ares *surv.*	rand. *surv.*	rand. *win*	finished	50
6.	rand. *surv.*	rand. *elim.*	rand. *elim.*	rand. *surv.*	Ares *surv.*	rand. *elim.*	Ares *win*	finished	58
7.	rand. *surv.*	rand. *elim.*	rand. *elim.*	rand. *elim.*	Ares *win*	Ares *surv.*	rand. *surv.*	finished	82
8.	rand. *elim.*	rand. *elim.*	rand. *elim.*	rand. *surv.*	Ares *surv.*	Ares *surv.*	Ares *win*	finished	52
9.	rand. *elim.*	rand. *elim.*	rand. *elim.*	Ares *surv.*	rand. *win*	rand. *elim.*	rand. *surv.*	finished	118
10.	rand. *elim.*	rand. *elim.*	rand. *surv.*	Ares *surv.*	rand. *surv.*	rand. *surv.*	Ares *win*	finished	42
11.	rand. *elim.*	rand. *elim.*	rand. *elim.*	Ares *win*	rand. *elim.*	Ares *surv.*	rand. *surv.*	finished	54
12.	rand. *elim.*	rand. *elim.*	rand. *surv.*	Ares *surv.*	rand. *surv.*	Ares *surv.*	Ares *win*	finished	34
13.	rand. *elim.*	rand. *elim.*	rand. *elim.*	Ares *surv.*	Ares *win*	rand. *elim.*	rand. *elim.*	finished	46
14.	rand. *elim.*	rand. *elim.*	rand. *surv.*	Ares *surv.*	Ares *surv.*	rand. *surv.*	Ares *win*	finished	36
15.	rand. *elim.*	rand. *elim.*	rand. *elim.*	Ares *surv.*	Ares *win*	Ares *surv.*	rand. *surv.*	finished	168
16.	rand. *elim.*	rand. *elim.*	rand. *surv.*	Ares *surv.*	Ares *surv.*	Ares *surv.*	Ares *win*	finished	54
17.	rand. *elim.*	rand. *draw*	Ares *draw*	rand. *elim.*	rand. *draw*	rand. *elim.*	rand. *draw*	pre-ended	218
18.	rand. *elim.*	rand. *elim.*	Ares *surv.*	rand. *elim.*	rand. *elim.*	rand. *elim.*	Ares *win*	finished	138
19.	rand. *elim.*	rand. *elim.*	Ares *draw*	rand. *elim.*	rand. *elim.*	Ares *draw*	rand. *elim.*	pre-ended	218
20.	rand. *surv.*	rand. *elim.*	Ares *surv.*	rand. *surv.*	rand. *surv.*	Ares *surv.*	Ares *win*	finished	48
21.	rand. *elim.*	rand. *elim.*	Ares *surv.*	rand. *elim.*	Ares *win*	rand. *surv.*	rand. *elim.*	finished	76
22.	rand. *elim.*	rand. *elim.*	Ares *surv.*	rand. *elim.*	Ares *win*	rand. *surv.*	Ares *surv.*	finished	98
23.	rand. *elim.*	rand. *elim.*	Ares *surv.*	rand. *elim.*	Ares *win*	Ares *surv.*	rand. *elim.*	finished	82
24.	rand. *elim.*	rand. *elim.*	Ares *win*	rand. *elim.*	Ares *surv.*	Ares *surv.*	Ares *surv.*	finished	188
25.	rand. *surv.*	rand. *elim.*	Ares *surv.*	Ares *win*	rand. *elim.*	rand. *surv.*	rand. *elim.*	finished	90
26.	rand. *elim.*	rand. *elim.*	Ares *surv.*	Ares *surv.*	rand. *surv.*	rand. *elim.*	Ares *win*	finished	152
27.	rand. *surv.*	rand. *elim.*	Ares *surv.*	Ares *win*	rand. *surv.*	Ares *surv.*	rand. *surv.*	finished	42
28.	rand. *elim.*	rand. *elim.*	Ares *surv.*	Ares *surv.*	rand. *surv.*	Ares *surv.*	Ares *win*	finished	68
29.	rand. *elim.*	rand. *elim.*	Ares *surv.*	Ares *elim.*	Ares *win*	rand. *elim.*	rand. *elim.*	finished	134
30.	rand. *elim.*	rand. *elim.*	Ares *surv.*	Ares *surv.*	Ares *surv.*	rand. *surv.*	Ares *win*	finished	86
31.	rand. *elim.*	rand. *elim.*	Ares *surv.*	Ares *surv.*	Ares *win*	Ares *elim.*	rand. *elim.*	finished	82
32.	rand. *elim.*	rand. *elim.*	Ares *surv.*	Ares *surv.*	Ares *surv.*	Ares *elim.*	Ares *win*	finished	198
33.	rand. *surv.*	Ares *win*	rand. *surv.*	rand. *surv.*	rand. *elim.*	rand. *surv.*	rand. *elim.*	finished	42
34.	rand. *elim.*	Ares *surv.*	rand. *elim.*	rand. *surv.*	rand. *elim.*	rand. *surv.*	Ares *win*	finished	70
35.	rand. *elim.*	Ares *win*	rand. *surv.*	rand. *surv.*	rand. *elim.*	Ares *surv.*	rand. *elim.*	finished	44
36.	rand. *elim.*	Ares *surv.*	rand. *elim.*	rand. *surv.*	rand. *surv.*	Ares *surv.*	Ares *win*	finished	36
37.	rand. *surv.*	Ares *win*	rand. *surv.*	rand. *elim.*	Ares *surv.*	rand. *elim.*	rand. *elim.*	finished	70
38.	rand. *elim.*	Ares *surv.*	rand. *elim.*	rand. *surv.*	Ares *surv.*	rand. *elim.*	Ares *win*	finished	42
39.	rand. *surv.*	Ares *elim.*	rand. *elim.*	rand. *elim.*	Ares *win*	Ares *surv.*	rand. *elim.*	finished	84
40.	rand. *surv.*	Ares *elim.*	rand. *elim.*	rand. *elim.*	Ares *surv.*	Ares *surv.*	Ares *win*	finished	110
41.	rand. *surv.*	Ares *win*	rand. *elim.*	Ares *surv.*	rand. *elim.*	rand. *surv.*	rand. *elim.*	finished	44
42.	rand. *elim.*	Ares *win*	rand. *surv.*	Ares *surv.*	rand. *elim.*	rand. *surv.*	Ares *surv.*	finished	44
43.	rand. *elim.*	Ares *win*	rand. *surv.*	Ares *surv.*	rand. *elim.*	Ares *surv.*	rand. *elim.*	finished	36
44.	rand. *elim.*	Ares *surv.*	rand. *elim.*	Ares *win*	rand. *elim.*	Ares *surv.*	Ares *surv.*	finished	118
45.	rand. *elim.*	Ares *win*	rand. *surv.*	Ares *surv.*	Ares *surv.*	rand. *surv.*	rand. *elim.*	finished	76
46.	rand. *elim.*	Ares *elim.*	rand. *elim.*	Ares *surv.*	Ares *surv.*	rand. *elim.*	Ares *win*	finished	76
47.	rand. *elim.*	Ares *win*	rand. *elim.*	Ares *surv.*	Ares *surv.*	Ares *elim.*	rand. *surv.*	finished	78
48.	rand. *elim.*	Ares *surv.*	rand. *surv.*	Ares *surv.*	Ares *surv.*	Ares *surv.*	Ares *win*	finished	44
49.	rand. *elim.*	Ares *win*	Ares *surv.*	rand. *elim.*	rand. *elim.*	rand. *surv.*	rand. *elim.*	finished	186
50.	rand. *surv.*	Ares *win*	Ares *surv.*	rand. *elim.*	rand. *elim.*	rand. *elim.*	Ares *surv.*	finished	74
51.	rand. *elim.*	Ares *win*	Ares *surv.*	rand. *elim.*	rand. *surv.*	Ares *surv.*	rand. *elim.*	finished	76
52.	rand. *elim.*	Ares *surv.*	Ares *win*	rand. *elim.*	rand. *elim.*	Ares *surv.*	Ares *elim.*	finished	114
53.	rand. *elim.*	Ares *win*	Ares *surv.*	rand. *elim.*	Ares *surv.*	rand. *surv.*	rand. *elim.*	finished	94
54.	rand. *elim.*	Ares *elim.*	Ares *elim.*	rand. *elim.*	Ares *surv.*	rand. *surv.*	Ares *win*	finished	34
55.	rand. *elim.*	Ares *win*	Ares *surv.*	rand. *elim.*	Ares *surv.*	Ares *surv.*	rand. *elim.*	finished	142
56.	rand. *elim.*	Ares *win*	Ares *surv.*	rand. *surv.*	Ares *surv.*	Ares *surv.*	Ares *elim.*	finished	106
57.	rand. *elim.*	Ares *win*	Ares *surv.*	Ares *surv.*	rand. *elim.*	rand. *surv.*	rand. *elim.*	finished	56
58.	rand. *elim.*	Ares *win*	Ares *surv.*	Ares *surv.*	rand. *elim.*	rand. *elim.*	Ares *elim.*	finished	102
59.	rand. *elim.*	Ares *draw*	Ares *draw*	Ares *draw*	rand. *elim.*	Ares *draw*	rand. *elim.*	pre-ended	218
60.	rand. *elim.*	Ares *draw*	Ares *draw*	Ares *draw*	rand. *elim.*	Ares *draw*	Ares *elim.*	pre-ended	218
61.	rand. *elim.*	Ares *win*	Ares *surv.*	Ares *surv.*	Ares *surv.*	rand. *elim.*	rand. *elim.*	finished	136
62.	rand. *elim.*	Ares *surv.*	Ares *elim.*	Ares *surv.*	Ares *surv.*	rand. *elim.*	Ares *win*	finished	88
63.	rand. *elim.*	Ares *draw*	Ares *draw*	Ares *draw*	Ares *draw*	Ares *draw*	rand. *elim.*	pre-ended	218
64.	rand. *elim.*	Ares *win*	Ares *surv.*	Ares *surv.*	Ares *surv.*	Ares *surv.*	Ares *surv.*	finished	116

game	Austria	Germany	Turkey	Italy	France	England	Russia	status	round
65.	Ares *win*	rand. *surv.*	rand. *surv.*	rand. *elim.*	rand. *surv.*	rand. *elim.*	rand. *elim.*	finished	50
66.	Ares *win*	rand. *elim.*	rand. *elim.*	rand. *elim.*	rand. *elim.*	rand. *surv.*	Ares *surv.*	finished	100
67.	Ares *win*	rand. *surv.*	rand. *elim.*	rand. *elim.*	rand. *surv.*	Ares *surv.*	rand. *surv.*	finished	78
68.	Ares *surv.*	rand. *elim.*	rand. *surv.*	rand. *surv.*	rand. *surv.*	Ares *surv.*	Ares *win*	finished	44
69.	Ares *win*	rand. *elim.*	rand. *surv.*	rand. *elim.*	Ares *surv.*	rand. *surv.*	rand. *surv.*	finished	46
70.	Ares *surv.*	rand. *elim.*	rand. *surv.*	rand. *elim.*	Ares *surv.*	rand. *elim.*	Ares *win*	finished	58
71.	Ares *win*	rand. *elim.*	rand. *elim.*	rand. *elim.*	Ares *surv.*	Ares *surv.*	rand. *elim.*	finished	204
72.	Ares *draw*	rand. *elim.*	rand. *elim.*	rand. *elim.*	Ares *draw*	Ares *draw*	Ares *draw*	pre-ended	218
73.	Ares *win*	rand. *elim.*	rand. *elim.*	Ares *surv.*	rand. *elim.*	rand. *surv.*	rand. *surv.*	finished	110
74.	Ares *surv.*	rand. *elim.*	rand. *elim.*	Ares *surv.*	rand. *surv.*	rand. *surv.*	Ares *win*	finished	54
75.	Ares *win*	rand. *elim.*	rand. *elim.*	Ares *surv.*	rand. *elim.*	Ares *surv.*	rand. *elim.*	finished	70
76.	Ares *surv.*	rand. *elim.*	rand. *elim.*	Ares *surv.*	rand. *surv.*	Ares *surv.*	Ares *win*	finished	36
77.	Ares *win*	rand. *elim.*	rand. *elim.*	Ares *elim.*	Ares *surv.*	rand. *surv.*	rand. *surv.*	finished	80
78.	Ares *surv.*	rand. *elim.*	rand. *elim.*	Ares *surv.*	Ares *surv.*	rand. *surv.*	Ares *win*	finished	36
79.	Ares *win*	rand. *elim.*	rand. *elim.*	Ares *elim.*	Ares *surv.*	Ares *surv.*	rand. *elim.*	finished	108
80.	Ares *surv.*	rand. *elim.*	rand. *elim.*	Ares *surv.*	Ares *surv.*	Ares *surv.*	Ares *win*	finished	60
81.	Ares *win*	rand. *elim.*	Ares *surv.*	rand. *elim.*	rand. *elim.*	rand. *surv.*	rand. *surv.*	finished	86
82.	Ares *win*	rand. *surv.*	Ares *surv.*	rand. *elim.*	rand. *surv.*	rand. *surv.*	Ares *elim.*	finished	60
83.	Ares *win*	rand. *elim.*	Ares *surv.*	rand. *elim.*	rand. *surv.*	Ares *surv.*	rand. *elim.*	finished	102
84.	Ares *win*	rand. *elim.*	Ares *elim.*	rand. *elim.*	rand. *surv.*	Ares *surv.*	Ares *surv.*	finished	68
85.	Ares *surv.*	rand. *elim.*	Ares *surv.*	rand. *elim.*	Ares *win*	rand. *surv.*	rand. *elim.*	finished	54
86.	Ares *draw*	rand. *elim.*	Ares *elim.*	rand. *elim.*	Ares *draw*	rand. *elim.*	Ares *draw*	pre-ended	218
87.	Ares *surv.*	rand. *elim.*	Ares *surv.*	rand. *elim.*	Ares *win*	Ares *surv.*	rand. *elim.*	finished	174
88.	Ares *draw*	rand. *elim.*	Ares *draw*	rand. *elim.*	Ares *draw*	Ares *draw*	Ares *draw*	pre-ended	218
89.	Ares *win*	rand. *surv.*	Ares *surv.*	Ares *surv.*	rand. *elim.*	rand. *surv.*	rand. *elim.*	finished	38
90.	Ares *elim.*	rand. *elim.*	Ares *surv.*	Ares *surv.*	rand. *surv.*	rand. *elim.*	Ares *win*	finished	40
91.	Ares *elim.*	rand. *surv.*	Ares *win*	Ares *surv.*	rand. *elim.*	Ares *surv.*	rand. *elim.*	finished	128
92.	Ares *surv.*	rand. *elim.*	Ares *surv.*	Ares *surv.*	rand. *elim.*	Ares *surv.*	Ares *win*	finished	110
93.	Ares *draw*	rand. *elim.*	Ares *draw*	Ares *elim.*	Ares *draw*	rand. *draw*	rand. *elim.*	pre-ended	218
94.	Ares *draw*	rand. *elim.*	Ares *draw*	Ares *elim.*	Ares *draw*	rand. *elim.*	Ares *draw*	pre-ended	218
95.	Ares *draw*	rand. *elim.*	Ares *draw*	Ares *draw*	Ares *draw*	Ares *draw*	rand. *elim.*	pre-ended	218
96.	Ares *draw*	rand. *elim.*	Ares *draw*	Ares *elim.*	Ares *draw*	Ares *draw*	Ares *draw*	pre-ended	218
97.	Ares *surv.*	Ares *win*	rand. *surv.*	rand. *elim.*	rand. *surv.*	rand. *elim.*	rand. *elim.*	finished	40
98.	Ares *surv.*	Ares *surv.*	rand. *elim.*	rand. *elim.*	rand. *surv.*	rand. *surv.*	Ares *win*	finished	80
99.	Ares *surv.*	Ares *win*	rand. *elim.*	rand. *elim.*	rand. *elim.*	Ares *surv.*	rand. *elim.*	finished	66
100.	Ares *elim.*	Ares *win*	rand. *surv.*	rand. *surv.*	rand. *elim.*	Ares *surv.*	Ares *surv.*	finished	36
101.	Ares *win*	Ares *surv.*	rand. *elim.*	rand. *elim.*	Ares *surv.*	rand. *surv.*	rand. *surv.*	finished	36
102.	Ares *draw*	Ares *draw*	rand. *elim.*	rand. *elim.*	Ares *draw*	rand. *elim.*	Ares *draw*	pre-ended	218
103.	Ares *draw*	Ares *draw*	rand. *elim.*	rand. *elim.*	Ares *draw*	Ares *draw*	rand. *elim.*	pre-ended	218
104.	Ares *win*	Ares *surv.*	rand. *elim.*	rand. *elim.*	Ares *surv.*	Ares *surv.*	Ares *surv.*	finished	62
105.	Ares *win*	Ares *surv.*	rand. *elim.*	Ares *surv.*	rand. *elim.*	rand. *surv.*	rand. *elim.*	finished	166
106.	Ares *win*	Ares *surv.*	rand. *elim.*	Ares *surv.*	rand. *elim.*	rand. *surv.*	Ares *surv.*	finished	58
107.	Ares *win*	Ares *surv.*	rand. *elim.*	Ares *surv.*	rand. *elim.*	Ares *surv.*	rand. *elim.*	finished	98
108.	Ares *surv.*	Ares *surv.*	rand. *elim.*	Ares *surv.*	rand. *elim.*	Ares *surv.*	Ares *win*	finished	66
109.	Ares *win*	Ares *surv.*	rand. *elim.*	Ares *surv.*	Ares *surv.*	rand. *elim.*	rand. *elim.*	finished	124
110.	Ares *surv.*	Ares *elim.*	rand. *elim.*	Ares *surv.*	Ares *surv.*	rand. *surv.*	Ares *win*	finished	78
111.	Ares *win*	Ares *surv.*	rand. *elim.*	Ares *elim.*	Ares *surv.*	Ares *surv.*	rand. *elim.*	finished	104
112.	Ares *elim.*	Ares *win*	rand. *elim.*	Ares *surv.*	Ares *surv.*	Ares *elim.*	Ares *surv.*	finished	74
113.	Ares *win*	Ares *surv.*	Ares *surv.*	rand. *elim.*	rand. *elim.*	rand. *elim.*	rand. *elim.*	finished	124
114.	Ares *surv.*	Ares *win*	Ares *surv.*	rand. *elim.*	rand. *elim.*	rand. *elim.*	Ares *elim.*	finished	188
115.	Ares *surv.*	Ares *win*	Ares *surv.*	rand. *surv.*	rand. *elim.*	Ares *surv.*	rand. *elim.*	finished	32
116.	Ares *draw*	Ares *draw*	Ares *draw*	rand. *elim.*	rand. *elim.*	Ares *draw*	Ares *elim.*	pre-ended	218
117.	Ares *draw*	Ares *draw*	Ares *elim.*	rand. *elim.*	Ares *draw*	rand. *elim.*	rand. *elim.*	pre-ended	218
118.	Ares *elim.*	Ares *surv.*	Ares *win*	rand. *elim.*	Ares *surv.*	rand. *surv.*	Ares *surv.*	finished	110
119.	Ares *draw*	Ares *draw*	Ares *draw*	rand. *elim.*	Ares *draw*	Ares *draw*	rand. *elim.*	pre-ended	218
120.	Ares *win*	Ares *elim.*	Ares *surv.*	rand. *elim.*	Ares *surv.*	Ares *surv.*	Ares *surv.*	finished	128
121.	Ares *surv.*	Ares *win*	Ares *surv.*	Ares *surv.*	rand. *elim.*	rand. *elim.*	rand. *elim.*	finished	120
122.	Ares *surv.*	Ares *win*	Ares *surv.*	Ares *elim.*	rand. *surv.*	rand. *elim.*	Ares *elim.*	finished	118
123.	Ares *win*	Ares *surv.*	Ares *surv.*	Ares *elim.*	rand. *surv.*	Ares *surv.*	rand. *elim.*	finished	56
124.	Ares *win*	Ares *surv.*	Ares *surv.*	Ares *elim.*	rand. *elim.*	Ares *surv.*	Ares *surv.*	finished	92
125.	Ares *elim.*	Ares *surv.*	Ares *surv.*	Ares *surv.*	Ares *win*	rand. *elim.*	rand. *elim.*	finished	128
126.	Ares *elim.*	Ares *draw*	Ares *draw*	Ares *draw*	Ares *draw*	rand. *elim.*	Ares *draw*	pre-ended	218
127.	Ares *elim.*	Ares *draw*	Ares *draw*	Ares *draw*	Ares *draw*	Ares *draw*	rand. *elim.*	pre-ended	218
128.	Ares *surv.*	Ares *elim.*	Ares *surv.*	Ares *surv.*	Ares *surv.*	Ares *surv.*	Ares *win*	finished	74

H Average progress per empire

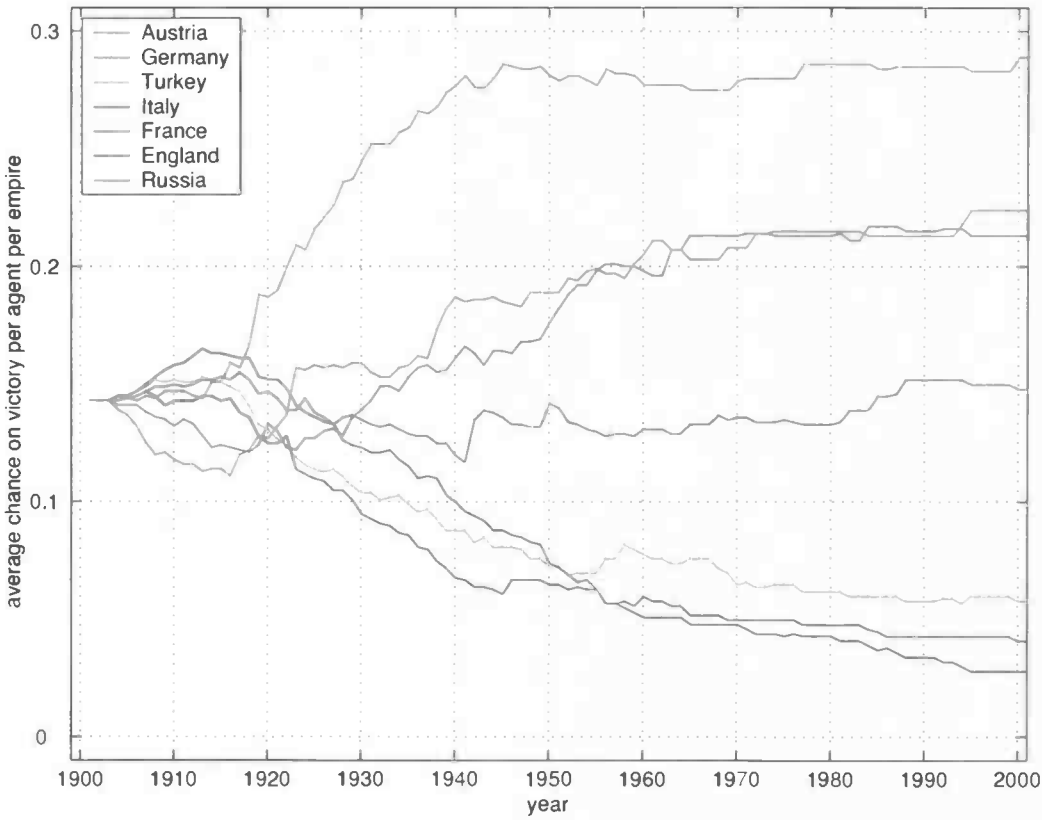

Figure 17: Average progress per empire