Towards General Cooperative Game Playing

João Marinheiro[1] and Henrique Lopes Cardoso[1,2]

[1] DEI, Faculdade de Engenharia, Universidade do Porto, Porto, Portugal
[2] LIACC – Laboratório de Inteligência Artificial e Ciência de Computadores, Porto, Portugal
joca.mar2011@gmail.com, hlc@fe.up.pt

Abstract. Attempts to develop generic approaches to game playing have been around for several years in the field of Artificial Intelligence. However, games that involve explicit cooperation among otherwise competitive players – *cooperative negotiation games* – have not been addressed by current approaches. Yet, such games provide a much richer set of features, related with social aspects of interactions, which make them appealing for envisioning real-world applications. This work proposes a generic agent architecture – *Alpha* – to tackle cooperative negotiation games, combining elements such as search strategies, negotiation, opponent modeling and trust management. The architecture is then validated in the context of two different games that fall in this category – Diplomacy and Werewolves. Alpha agents are tested in several scenarios, against other state-of-the-art agents. Besides highlighting the promising performance of the agents, the role of each architectural component in each game is assessed.

Keywords: Multi-Agent Systems, Cooperative Games, General Game Playing, Negotiation, Strategy, Opponent Modeling

1 Introduction

From the beginning of AI research, games have been an important test-bed to develop new and interesting strategies and models for a variety of applications. While there has been extensive research using a number of different games, most work in this area relates to specific individual games. As a consequence, agents developed for one game are often difficult to adapt to other games due to the use of game-specific heuristics and architectures. One field of research that has attempted to mitigate this problem is that of general game playing [12, 19]. General game playing agents provide useful insight into what are the essential elements for AI to emulate human thought and adapt to new situations never encountered before.

Most existing work in the field of game-playing agents relates to traditional adversarial games like Chess [9] or Go [20], because of their simple rules and large strategic depth. These kinds of games, while useful in many regards, do not provide the best environment to allow modeling more complex and interesting social interactions between players. One interesting category of games that allows

social interactions is that of *cooperative negotiation games*, where players are encouraged to barter and create or break deals between themselves in order to obtain the best results in the game.

Targeting general game playing AI for cooperative games would allow for the development of increasingly interesting and complex agent capabilities and features, enabling agents to not only adapt their playing strategies to different games but also negotiate in a variety of different environments with different protocols, goals and issues. In turn, this would allow to address increasingly complex and interesting real-world scenarios, a concern that should always be in place when doing research in games. To aspire such an aim, it is important to first determine the essential elements that agents must have in order to be able to play cooperative negotiation games effectively. Towards that direction, this work attempts to identify some of these elements and propose a generic agent architecture that tackles them and facilitates the development of agents with such capabilities, able to play cooperative negotiation games.

The rest of this paper is structured as follows. Section 2 provides insight into cooperative games, including their characteristics and challenges. In Section 3 we revise some of the existing approaches to generic game playing and to cooperative multi-agent games in particular. Section 4 introduces a general architecture for cooperative negotiation games, including a description of each of its modules, and a brief description of a general framework implementing the architecture. Then, in Section 5, we describe implementations of the meta-model described in Section 4, delivering some agents for two cooperative negotiation games. Section 6 reports on experimental evaluation of the developed agents, which have been tested against state-of-the-art agents, when available. Section 7 puts the contributions of this paper in the perspective of a long-term goal of delivering general cooperative game playing agents. Finally, in Section 8 conclusions are drawn and avenues of future work are laid out.

2 Cooperative Multi-Agent Games

Traditionally in the field of game theory, a game is considered a cooperative game [15] if players are able to form binding commitments with each other. Games in which players cannot create binding agreements are then considered non-cooperative games. It is usually assumed that communication between players is allowed in cooperative games but not in non-cooperative games.

For the purposes of this work however, we consider a somewhat more general definition of cooperative games – games in which cooperation between players is possible and encouraged but in which binding agreements are not necessarily prevalent. More specifically, we will focus on cooperative negotiation games with a *mix of cooperation and competition*. In this setting, negotiation is used to establish cooperation in specific phases of an otherwise competitive game. Some characteristics that are frequently present in these games are:

– Very large search spaces, which makes the application of traditional search techniques impractical.

– Difficulty in evaluating moves and player positions due to the fact that evaluating such moves often depends on the *social context* of the game.
– The possibility of betrayals and desertions due to the existence of non-binding agreements.

There are many kinds of cooperative negotiation games, such as *Settlers of Catan*, *Quo Vadis?* or *Genoa*. In the context of this work, two very relevant cooperative negotiation games are *Diplomacy* and *Werewolves of Miller's Hollow*.

Diplomacy [3] is a strategy game for 7 players where each player takes control of a nation and their armies and navies. By submitting orders to these units, which are executed simultaneously with those of other nations, players attempt to capture territories and hold supply centers in a map of Europe. The first player to capture 18 or more supply centers wins the game. Players can also issue orders to support the orders of other players in the game, and they are encouraged to negotiate among each other in order to form alliances, support each other and create joint plans. Because of this the game highly encourages cooperation and players that are able to effectively negotiate are able to obtain much better results in the game.

Werewolves of Miller's Hollow [8] is a team-based game where two teams – the villagers and the werewolves – attempt to eliminate each other. The twist is that players on the villager team do not know who their allies are and who the werewolves are, encouraging players to communicate, share information and decide who to trust. The game is played in two phases: the day phase where players communicate freely, which ends with a vote to eliminate one player who the remaining players think might be a werewolf, and the night phase where players may not communicate but can choose to use some special abilities depending on their role in the game.

Both Diplomacy and Werewolves of Miller's Hollow are negotiation games with a mix of cooperation and competition that allow players to communicate among themselves in order to reach non-binding agreements, and use their actions to support or hinder each other. Both are deterministic and imperfect information games that work by phases. Despite these similarities, however, they are games with very different features. While Diplomacy is for the most part a zero-sum game, Werewolves of Miller's Hollow is not, with several players often losing or gaining utility with certain actions. While Diplomacy is a competitive game by nature where each player is ultimately hoping to be the one to win the game itself, which encourages eventual betrayal even among long time allies, Werewolves of Miller's Hollow is a cooperative game where players are divided into teams and encouraged to cooperate among themselves – agents win or lose the game as a group. The difference to a typical team based game is that in Werewolves of Miller's Hollow players do not have complete information about who is on their team and who is on the opposing team, and must thus be cautious about who they choose to trust. Table 1 summarizes a comparison of the characteristics of Diplomacy and Werewolves of Miller's Hollow.

Due to the characteristics of cooperative negotiation games, it is possible to obtain much better results in these games if one is able to negotiate and

Table 1. Comparison between Diplomacy and Werewolves of Miller's Hollow

Feature	Diplomacy	Werewolves
Deterministic	Yes	Yes
Information	Imperfect	Imperfect
Zero-Sum	Yes	No
Non-binding agreements possible	Yes	Yes
Simultaneous moves	Yes	Yes
Communication	Public & Private	Mostly public
Player victory	Individual	Team

coordinate with other players effectively. Unfortunately, while humans are very good at negotiation and intuitively know who to trust, it is much harder for a computer to do so. In order to develop effective and believable AIs for this sort of games, and cope with the large size of their search spaces, new strategies that can effectively combine search strategies with negotiation and opponent modeling need to be employed.

3 Related Work

There are relevant works both in the area of general game playing and cooperative negotiation games. We here do not aim to provide an exhaustive list of game playing agents, but instead to discuss some of the most important approaches to general game playing and to cooperative negotiation games.

There have been several attempts to develop generic game playing agents that are able to understand and play a variety of games. These systems usually require a set of rules and constraints that formalize how the game is played, usually defined in a specific game description language.

Zillions of Games is a system developed by Mallet and Leffer [4], where programmers can define a wide variety of two dimensional abstract board games using rule files written in a proprietary language (ZRF files). The platform can then read these files and generate the game as well as intelligent general AIs that can play it. While the system is limited in the rules and layouts of the games it can generate (and its AI lacks advanced features such as negotiation or trust reasoning capabilities), it is nevertheless an interesting example of a general AI that is adaptable enough to play in a variety of different environments.

One of the most well known projects in this field is Stanford University's General Game Playing (GGP) project [12, 19]. This project provides a framework upon which developers can create general game playing agents to play a variety of games, as well as the tools to describe those games. Games are described in a description language called GDL, which is then interpreted by the agents developed using the GGP framework. While GGP is mostly focused towards traditional board games a similar project by the University of Essex, the General Video Game AI (GVGAI), is more focused towards computer video games [17]. Similarly to GGP it provides a game description language, VGDL,

and a framework upon which agents can be built, that are able to interpret and play any video game described using that language.

An approach to handle a particular family of imperfect information games is the Poker Game Description Language (PGDL) [5], focused in Poker and its many variants. PGDL is a language that allows users to define any Poker variant. Additionally, the PokerLang [18] high-level language facilitates the specification of agent strategies and tactics for specifying Poker playing agents.

Finally, more recently, Google's DeepMind project aims to apply deep learning techniques to a variety of scenarios, including games. Using neural networks, agents have been created that are able to effectively play games which contain extremely large search spaces, such as Go [20]. This deep learning approach can be applied to a variety of games and scenarios as long as one has access to quality training data with which to train the agent.

In the area of cooperative negotiation games there have also been several agents developed that provide an interesting starting point. Most of the work in this area is focused on the classic game of Diplomacy. Unfortunately, while there exist many agents for Diplomacy, no existing approaches have been found for the game of Werewolves of Miller's Hollow.

DumbBot is one of the simplest existing Diplomacy playing agents and was developed by Norman [16]. This bot has no negotiation capabilities and uses a simple heuristic to decide its actions by preferring to choose moves that weaken its strongest opponents. DumbBot assigns a value to each territory that depends on who controls it and what units are around it, and then it assigns actions to every unit depending on those score values. While the method used is very simple, DumbBot obtains fairly good results and is frequently used as a benchmark for other Diplomacy agents.

One of the most important and influential negotiating agents is the Israeli Diplomat. The architecture of the Israeli Diplomat was designed to be a general negotiation architecture to be applied in a variety of situations. This architecture was used to create a Diplomacy playing agent [13]. The Israeli Diplomat tries to mimic the structure of a war-time nation. It consists of several components working together to choose the best course of action. These components are the Prime Minister, the Ministry of Defense, the Foreign Office, the Military Headquarters, Intelligence and the Strategies Finder. The diplomat keeps a knowledge-base of the relations it believes each nation has with each other as well as any agreements it has, its intention to keep them and its trust that others will keep them. This knowledge-base is updated by the different modules as the game progresses, and affects every decision that the diplomat takes.

Another interesting approach is that of D-Brane [6], an agent developed by de Jonge that makes use of the NB^3 algorithm [7] as well as a complex strategical module to find the best sets of moves to negotiate and play. NB^3 is based on the branch-and-bound search algorithm and mixes negotiation with search, so that the former can direct the latter. It was designed to be used in environments where the search space is too large for traditional search techniques to be employed, and where better solutions often require cooperation among several agents. D-

Brane uses a basic form of opponent modeling by using the utility values of deals previously proposed and accepted by its opponents as a way to direct the search for better solutions; however, it does not make an attempt to explicitly predict an opponent's goals or strategies. Another aspect that this agent lacks is the ability to negotiate coalitions with other agents as well as joint moves for future phases of the game, having no negotiation strategy for these kinds of deals.

DipBlue [10] is another negotiating agent for Diplomacy, inspired by the Israeli Diplomat architecture. DipBlue is split into several modules called Advisers, that together decide the actions the agent takes. Each adviser receives the evaluated move scores from previous advisers and alters them according to its role. The base adviser is inspired by DumbBot and uses the same scoring heuristic. This score is then changed by other advisers to promote support actions for the units, promote actions that keep agreements with its allies and encourage the agent to attack players that it distrusts. In order to model the trust value of each player, DipBlue keeps a trust matrix that is updated as the game is played [11]. If a player performs hostile actions against DipBlue, such as attacking it or breaking an agreement, its trust value diminishes. If a player performs friendly actions, or refrains from doing hostile actions, its trust value increases. DipBlue is more likely to accept agreements and help players with which it has a high trust value, and attack players with a low trust value. DipBlue does not have full negotiation capabilities, lacking the ability to ask and give information or threaten players.

Concerning other cooperative negotiation games, perhaps one of the more interesting approaches is the work by Afiouni and Øvrelid [1] that builds upon the opponent modeling techniques described by Krimpen *et al.* In their work, they propose a negotiating agent that uses weighted constraints to evaluate offers. By watching the variation in issues in offers proposed by its opponents, this agent can add or remove constraints from its opponent model, or alter their weights. It then proposes new offers by solving a prioritized constraint satisfaction problem.

4 A General Architecture for Cooperative Negotiation Games

A general architecture for cooperative negotiation games is needed to address the goal of obtaining general cooperative game playing. With this in mind, in this section we describe the *Alpha architecture* (first introduced in [14]) and its modules. This architecture enables approaching several of the challenges present in cooperative negotiation games, and facilitates the development of agents to play these games effectively.

4.1 The Alpha Architecture

In order to address the development of agents for cooperative negotiation games, we need to take into consideration several complex issues, such as negotiation abilities, opponent modeling and trust relationships. Making the architecture

generic and game-independent will ensure its applicability in a wide range of cooperative negotiation games. As such, no assumptions should be made regarding the usage of specific negotiation protocols or strategies, and the formalization of goals and search strategies should also be game-agnostic. The *Alpha architecture* is modular and based on the architectures of Israeli Diplomat [13] and DipBlue [11]. Figure 1 shows a simplified overview of the Alpha architecture.

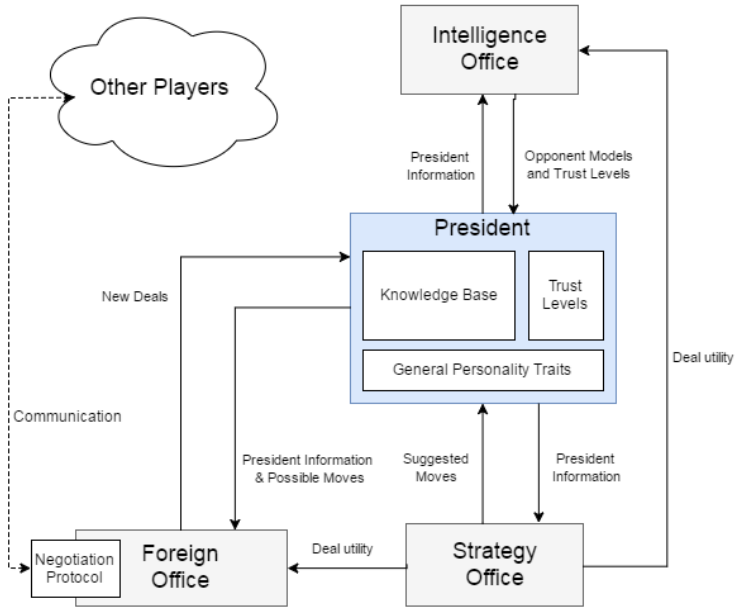

Fig. 1. High-level diagram of the Alpha architecture and its modules.

The architecture has a structure similar to the Israeli Diplomat, with four independent modules:

- The **President** coordinates the interactions between all modules and is responsible for the final decisions regarding game play.
- The **Strategy Office** is in charge of suggesting good strategies to the President.
- The **Foreign Office** deals with negotiation with other players in the game.
- The **Intelligence Office** makes predictions regarding what opponents are likely to do in the game.

This structure allows the architecture to have a clean separation between different independent modules that deal with different issues: negotiation, opponent modeling, strategic/tactical evaluation of the game, which are then combined with high-level agent personality and overall strategy. The idea is to enable and facilitate compositionality when building agents with specific characteristics, by combining these modules in different ways.

The President. The agent's central module is the President (PR), which holds its personality characteristics. This module coordinates interaction with the other modules, defining the overall high-level strategy of the agent through the definition of its goals. The PR is in charge of selecting and executing the agent's moves in the game. Figure 2 shows a simplified overview of the components of the PR module.

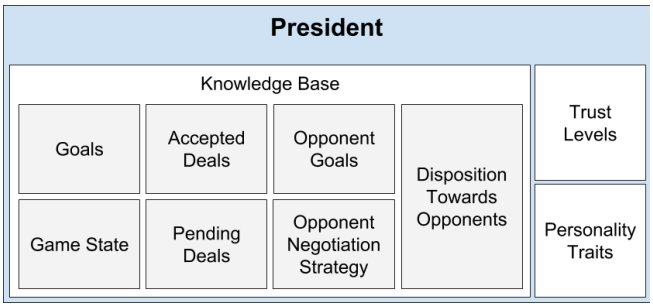

Fig. 2. High-level diagram of the President's components.

The PR keeps a knowledge-base including information about the game environment and its opponents. This knowledge-base is used and modified by the remaining three modules, allowing the PR to make decisions with up-to-date information regarding the environment. The information contained in the knowledge-base is the following:

- The current state of the game.
- The moves played during the course of the game by each player.
- The agent's current goals and their importance in the game.
- The lists of confirmed, completed and proposed deals by the agent and other players over the course of the game.
- The player's current disposition towards other players, such as who are its allies and enemies.
- The opponent models for each other player.
- The general trustworthiness levels of each player.
- The trustworthiness levels of each deal.

Since the different modules share and use a lot of the same information, one design concern was simplifying the sharing and exchange of information among the different modules, which lead to the decision to include this central knowledge-base in the PR module, which is easily accessible to all other modules. This also allows for the President to override decisions taken by its subordinate modules, such as which deals are confirmed.

Additionally, the PR includes a set of personality traits that can be defined depending on the game being played. These govern the general strategy of the

PR, such as how aggressive it is, how trusting of other players it is or how prone to taking risks it is. Finally, the PR also keeps lists of moves and deals suggested by the other modules, which the PR can then choose to execute depending on several factors.

When defining the PR module, the developer must define what constitutes a deal, a move and a goal, as these are game-specific concepts. Different games have very different possibilities for what moves and deals a player can make. Instances of these notions specify the game-specific elements that are stored in the knowledge-base and used by all modules.

One important role of the President is deciding what overall goals the agent is striving for and what relative importance to attribute to each goal. This information is passed on to other modules, which take it into consideration in their reasoning processes. This approach allows the PR to dictate the overall strategy it wants to follow to its subordinate modules, allowing them to focus on the individual details of what actions are more likely to be effective in attaining these goals. Different PR modules allow the developer to customize the agent's general strategy and personality, allowing for different player archetypes.

The Strategy Office. A strategical assessment of the game is conducted by the Strategy Office (SO), which has the responsibility of suggesting appropriate moves to the PR. The SO contains most of the game-specific heuristics, and evaluates the utility of possible moves and deals. For this reason, the SO is highly dependent and adapted to the specific game being played. It also defines the search strategy used to explore the space of possible moves.

The SO is conceptually split into two parts: the *search strategy* and the *evaluation method*. The search strategy is used for exploring the space of possible moves, while the evaluation method is used to assess the utility of moves and deals. While the search strategy can, in principle, be applied to different environments with a low amount of effort, tactical evaluation is, in general, entirely dependent on the game being played, as it relies on specific knowledge about the game's rules.

In order to find the best moves, the SO has access to the PR's knowledge-base, which includes information that is relevant to evaluate the utility of different moves and deals. The PR requests move suggestions to the SO, which replies with moves that are then stored in the PR's internal list.

Changing the SO amounts to choosing among different search strategies and heuristics for the game, which can have a major impact on a player's effectiveness.

The PR and SO modules were conceived because of a design concern with separating the long term "macro" strategy and the short term "micro" strategy. This separation, in the form of the PR and SO (and also the FO when it concerns negotiation), is useful since it allows for the agent to tackle complex problems more easily, by dividing them into smaller problems. The SO and FO only need to worry about a small subset of the overall game at a time, which simplifies the development of these agents, while also allowing the agent to be formulated

through long term plans without having to concern itself directly with more minor details of how to execute them via the PR.

The Foreign Office. The purpose of the Foreign Office (FO) is to manage any interaction with other players and negotiating deals and coalitions in a way that best allows the PR to execute the moves it is considering. By sending a list of moves, the PR requests the FO to find supporting deals through negotiation with other players. Also using any other information available in the PR's knowledge-base, the FO autonomously communicates with other players and decides what deals to propose, reject and accept. When a deal is proposed, confirmed or completed the FO informs the PR so that these deals are appropriately stored in its knowledge-base.

The FO includes a Negotiation Strategy that determines what deals are proposed and accepted and what concessions the agent is willing to make. This module also defines the negotiation protocol used by the agent when communicating with other players. The decision of what protocol to use is often dependent on the game being played or even the specific development framework on top of which the agent is being implemented.

By changing the FO, a developer can customize the negotiation capabilities of the agent, allowing the use of different negotiation and concession strategies. Omitting this module altogether leads to having an agent with no negotiation capabilities.

Having all negotiation handled by the FO and all actions performed by the PR allows the architecture to be cleaner, by compartmentalizing all platform specific code for interacting with the game environment and other agents in these two modules. It also more easily allows the agent to concurrently negotiate agreements with its peers, while simultaneously considering and possibly executing other incompatible moves and deals.

The Intelligence Office. Collecting information about and building models of opponents is an important aspect of games, and particularly in those involving social interactions. The purpose of the Intelligence Office (IO) is to address these needs by calculating trust values and building opponent models for the different players in the game.

The IO is divided into two parts: the opponent modeling function and the trust reasoning function. The opponent modeling function outputs the predicted goals and their relative importance for each opponent, to be updated in the PR's knowledge-base. How this is done is often specific to each game, since the goals themselves as well as the actions and deals being analyzed are also game-specific. Similarly, the trust reasoning function outputs trustworthiness values both for each opponent as well as for each individual deal, depending on how likely they are to be kept.

Configuring the IO can allow the developer to customize the opponent modeling and trust reasoning strategies, or lack thereof, of an agent. This module

is especially useful in conjunction with the FO, since negotiations are likely to benefit from a good opponent model and accurate trust reasoning.

The design decision of separating opponent modeling and trust reasoning in the IO is related to the fact that while both subjects deal with predicting what actions an opponent will do, they are independent of each other. For example, one can define an agent with no trust reasoning capabilities if this agent is not expected to negotiate, since trust reasoning is especially important for negotiation, while still using opponent modeling. Opponent modeling is useful regardless of whether negotiation is possible, since predicting what goals enemy players might have can have an impact on the tactics and strategy of the SO and PR.

4.2 The Alpha Framework

In order to facilitate the application of the Alpha architecture, a simple Java framework was developed, composed of several abstract classes representing each of the described modules and their behavior. These classes define what each module should do as well as the data that they can access and how this data is updated and communicated to and by each of the modules. Each module is defined in its own class and implements a specific interface available to every other module. In addition, there are several data classes that contain data relative to the game being played and the agent itself, such as the knowledge-base or the agent's personality traits.

In order to implement an agent using the Alpha framework, a developer has to implement the abstract classes of the modules he wishes to use. When implementing the modules, the developer must implement their abstract methods in order to define the domain-specific negotiation strategies, protocols, heuristics, models and message handling. In the PR, the developer defines the high-level strategy for the agent, such as how it decides which goals are more important and what disposition it has towards other players. Optionally, the developer can also specify actions for the agent to take before and after playing, such as initializing or cleaning up data. In the SO, the developer implements utility functions for moves and deals, as well as the search strategy used to find and suggest moves to the PR. In the FO, the negotiation protocol and strategies are implemented as well as how the agent sends and receives domain-specific negotiation messages. Finally, in the IO there are functions where the developer may implement trust reasoning and opponent modeling strategies.

Data produced by the different modules is automatically updated and made available to the relevant modules. To make use of such data (e.g. use opponent goal predictions, calculated by the IO, in order to enhance negotiations in the FO), a developer has access to it in the PR's knowledge-base.

The developer must also implement data classes with domain-specific definitions of moves, deals, goals and opponents. Having all of this set up, the different modules can be attached to the PR and the agent be made to play the game.

This framework is publicly available at: *https://github.com/jocamar/Alpha-Architecture*.

5 Building Agents with the Alpha Architecture

We here describe our experience on making use of the Alpha architecture, by implementing two agents for two very different cooperative negotiation games: Diplomacy and Werewolves of Miller's Hollow. These agents, developed for such different games, can be seen as proof of concept to test the Alpha architecture in distinct scenarios, each with its own challenges.

5.1 AlphaDip

AlphaDip is a Diplomacy playing agent heavily based on D-Brane [6], using a modified version of its strategic module as well as the NB^3 algorithm to search for the best moves. It has a few key improvements compared with D-Brane, the most notable ones being an improved strategic module, a defined strategy for negotiating coalitions and some ability to predict opponent goals and trustworthiness. A high-level diagram showing how the Alpha architecture was applied when building AlphaDip is shown in Figure 3.

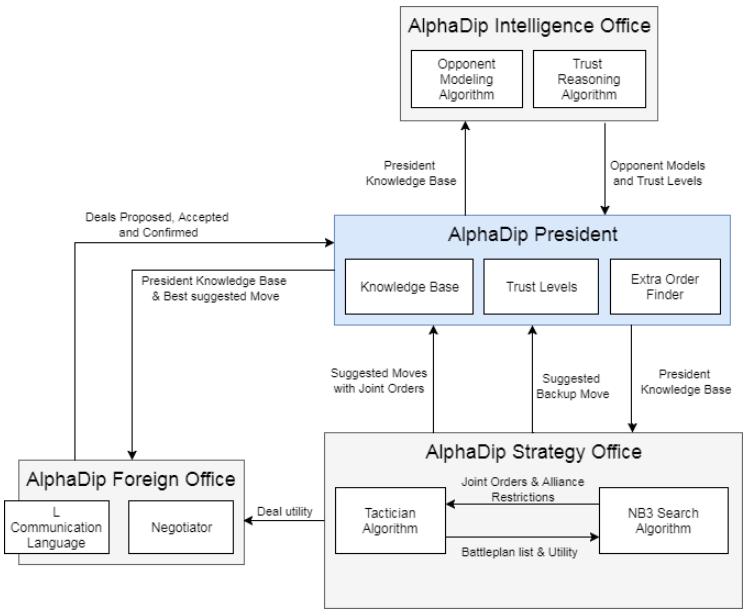

Fig. 3. AlphaDip's architecture.

The President. As detailed previously, the agent's PR is in charge of dealing with the high-level strategy for the agent and holds a variety of information such as player goals, moves to execute and trust levels. In order to discuss AlphaDip's

implementation it is necessary to understand how these concepts are represented by the PR, and consequently, understood by the remaining modules.

In the context of AlphaDip, a move is a set of orders, one for each unit the player controls. In Diplomacy, winning the game amounts to having the goal of capturing as many supply centers as possible – this translates into modeling a player's goals as how much they want to control each supply center in the current game stage. We model this as positive or null real numbers, where 0 means no intention to control a supply center, 1 means neutral intention to control a supply center and greater values mean greater intentions of controlling a supply center. On the other hand, trust values for players stored by the PR are positive real numbers that are inversely proportional to the trustworthiness of these players: 0 represents full trust in a player, 1 represents neutral trust and greater values indicate lower levels of trust in a player.

The PR's main role is coordinating the remaining modules and executing moves. To fulfill this role, the PR first asks the SO for a fall-back move When a round starts: this is a move that is expected to work even without any supporting deals from other players nor any kind of negotiation, and will be used by the PR in case all negotiations fail, or in the absence of a FO. Afterwards, the PR periodically asks the SO, who continually searches for the best moves, for suggested moves to consider. As these moves are discovered by the SO, the PR forwards the last, and currently best known, move on to the FO to negotiate for any required deals. After a certain time has elapsed, if all of the prerequisite deals have been confirmed by the FO the PR selects the last move suggested by the SO. Otherwise it falls back onto the fallback move calculated at the start.

The Strategy Office. AlphaDip's SO tries to find moves that maximize the number of controlled supply centers, and is based on D-Brane's strategic module and the NB^3 algorithm. The objective of the game is to take control of as many supply centers as possible. As such, one way of determining the utility for a move is simply the number of supply centers that are ensured to be controlled by a player when it plays that move. This is how D-Brane calculates the utility of a move. AlphaDip calculates utility in a similar way, but introduces trust reasoning and the prediction of opponent goals in order to attempt to obtain a more accurate value than D-Brane. While D-Brane attributes the same score of 1 to every supply center, AlphaDip uses its goals to influence the value of each supply center. Additionally, if a move requires supporting orders from other players to succeed, their trust values are taken into account when determining the utility of the move. Using these computations, the SO suggests moves that are likely to be easy for the FO to obtain any necessary supporting move commitments from other players. Equation 1 shows how the SO determines the utility $U_p(m)$ of a move m for player p, where n is the total number of supply centers. Function I_m returns 1 if the supply center i is sure to be controlled after move m and 0 if not. Function g_p is the goal value that player p has or is assumed to have for each supply center. Finally, t_p is the average trust that player p has on all other players involved in the move or 1 if no other players are involved.

$$U_p(m) = \frac{\sum_{i=1}^{n} I_m(i) \times g_p(i)}{t_p(m)} \quad (1)$$

In order to find the best moves in a given round, the SO is divided into two components: the *Tactician* and the *Searcher*. The Tactician attempts to find the best set of orders taking into account certain constraints such as any existing order commitments. It then calculates the utility of the move as described above. The Tactician uses a similar method to D-Brane's strategic module [6]. This method, exemplified in Algorithm 1, splits the current round into several smaller battles for individual supply centers. For a given set of orders that attempt to capture a supply center – a battle-plan – it is simple to calculate whether the supply center will be captured. We do this by comparing it with every possible enemy battle-plan for that supply center (lines 9 and 11 in Algorithm 1). If a battle-plan ensures the capture or defense of a supply center, we say that it is an invincible battle-plan. We can also determine pairs of invincible battle-plans, that is, two battle-plans that if executed simultaneously guarantee that at least one of them succeeds (lines 16-19). After the Tactician has found the existing invincible battle-plans and pairs of battle-plans, an And-Or search is employed to find the largest set of compatible invincible battle-plans or invincible pairs, that are also compatible with any existing order commitments (line 25).

The Searcher uses the NB^3 algorithm to look for joint moves with other players that can maximize the utility for all players involved. Each node in the search tree is a joint battle-plan with one or more opponents that attempts to capture a supply center or help another player capture it. The path from a node in the tree to the root of the tree contains a set of commitments that the Tactician will attempt to solve for in order to find the best possible compatible orders for the remaining units and the subsequent utility value for that node. If an acceptable move is found, these commitments would then have to be negotiated by the FO with any other involved players, for the move to be able to be executed. By looking for joint moves, the SO can find strategies and moves that would not be possible if a player was acting completely independently.

Because the SO uses the trust values and predicted opponent goals when calculating the utility of a node, the search of the NB^3 algorithm will be directed towards nodes with joint moves with players in whom AlphaDip trusts, and who are more likely to accept the conditions of the joint commitments.

The Foreign Office. Since, in order to be successfully executed, the moves suggested by the SO may include order commitments with other players, the PR passes the current best move it is considering on to the FO for it to negotiate any required support agreements. AlphaDip's FO performs two types of negotiation: coalition establishment with other players and order commitments for the current round. This is an improvement over D-Brane, which did not have a strategy for the establishment of coalitions, instead assuming that all D-Branes simply

Algorithm 1 Tactician Algorithm

```
1: playerAndOpponentBattleplans ← calculateBattleplans(gameState, player, allies)
2: playerAgreements ← getAgreementsFromPR()
3: for all bp ∈ playerAndOpponentBattleplans do
4:    if !isCompatible(bp,playerAgreements) then
5:       Remove bp from playerAndOpponentBattleplans
6:    end if
7: end for
8: playerPlans ← getPlayerPlans(playerAndOpponentBattleplans)
9: opponentPlans ← getOpponentPlans(playerAndOpponentBattleplans)
10: for all bp ∈ playerPlans do
11:    if !hasDefeatingPlans(bp, opponentBattleplans) then
12:       Add bp to invinciblePlans
13:    else
14:       defeatingPlans ← getDefeatingPlans(bp, opponentBattleplans)
15:       for all bp2 ∈ playerPlans do
16:          if isCompatible(bp,bp2) then
17:             defeatingPlans2 ← getDefeatingPlans(bp2, opponentBattleplans)
18:             if !hasLegalCombinationOfPlans(defeatingPlans,defeatingPlans2) then
19:                Add bp and bp2 as a new pair to invinciblePairs
20:             end if
21:          end if
22:       end for
23:    end if
24: end for
25: bestBattleplans ← getBestCombAndOrSearch(invinciblePlans,invinciblePairs)
26: return bestBattleplans
```

formed a coalition against all other players in the game. Currently, AlphaDip is not able to negotiate move commitments for the following rounds as that would increase the complexity of the agent tremendously.

The strategy employed to negotiate coalitions is similar to DipBlue's [11]. At the start of the game, AlphaDip proposes a peace agreement to every other player in the game. After that, during the rest of the game the FO attempts to propose alliances against the stronger player in the game with which it is not in peace with. If a player's trust value rises above a certain threshold (meaning the player is less trusted) the peace with that player is broken. Conversely, if the trust value drops below a certain level (meaning the player is trusted) AlphaDip proposes peace to this player. Additionally, if the game has 4 or less players remaining, AlphaDip immediately breaks any alliances it has with a player if that player controls 14 or more supply centers. This is so that AlphaDip does not let a player get too close to winning in the final stages of the game.

The FO also attempts to negotiate joint order commitments for the current round. The PR periodically asks the FO to negotiate deals concerning the moves being currently considered. The FO compares the utility of the suggested moves with the utility of each of the proposals it received: the FO either accepts the

best proposal received if it has more utility, or proposes any necessary joint order commitments required by the moves proposed by the PR.

In case the FO receives a proposal that is compatible with any deals it has already accepted, the FO asks the SO to calculate the utility of that deal, and informs the PR so that it stores the deal in the proposed deals list. The reason it does not choose to immediately accept or reject the proposal, as explained in [6], is so that the SO is given some time to continue searching and looking for any possibly better options for other joint moves, before the agent commits to the proposed orders. By committing to an offer and adding it to the PR's confirmed deals list, the search performed by the SO is automatically constrained to only look for moves that satisfy the conditions in the accepted deals.

The Intelligence Office. AlphaDip may use the IO to calculate trust values for players and predict their current goals in the game. In order to update the trust values, the IO uses a strategy similar to DipBlue [10], where trust in players increases steadily over the course of the game if no aggressive actions are taken by these players, and decreases when aggressive actions are taken. The magnitude of these updates is dependent on current trust values associated with the players, as well as whether AlphaDip considers himself to be at peace or at war with them. This way, if a player is highly trusted or in peace with AlphaDip, any aggressive actions it takes will have a bigger impact on that player's trust. On the other hand, if a player is not trusted or is at war with AlphaDip, any aggressive actions it takes have a smaller impact on that player's trust, since AlphaDip already expects that player to take aggressive actions.

The IO also attempts to predict its opponents' goals, that is, which supply centers it believes each player wants to control more, using a simple strategy exemplified in Algorithm 2. Each time a player takes an offensive action against a certain supply center, the IO increases the likelihood that that player wants to control that supply center (line 9 in Algorithm 2). If a player takes no offensive actions against a supply center, or takes actions that would help another player capture that supply center (such as support orders), the IO decreases the likelihood that the player wants to control that supply center (lines 5 and 12). Similarly to trust value updates, the magnitude of such increases and decreases are influenced by the current values for each supply center. This way, if a player is already expected to want control of a certain supply center, any actions it takes have a small impact on the value for that supply center desire. On the other hand, if a player suddenly makes a move on a supply center that AlphaDip believed that player was not interested in, the value for that supply center will be affected more significantly – this can be seen as an ability of AlphaDip in detecting changes in opponents' goals.

5.2 DipBlue

In addition to AlphaDip, we have made a reimplementation of DipBlue in light of the Alpha architecture (see Figure 4). In order to do this we split the DipBlue

Algorithm 2 AlphaDip IO Goal Prediction Algorithm

```
playerOpponents ← getOpponentsFromPR()
opponentGoals ← getOpponentGoalsFromPR()
for all op ∈ playerOpponents do
    for all g ∈ opponentGoals[op] do
        g ← g × 0.99
        actionsSupportingGoal ← getActionsSupportingGoal(op,g)
        actionsAgainstGoal ← getActionsAgainstGoal(op,g)
        for all o ∈ ordersAndDealsSupportingGoal do
            g ← g + 0.04/g
        end for
        for all o ∈ ordersAndDealsAgainstGoal do
            g ← g × 0.95
        end for
    end for
end for
return opponentGoals
```

architecture into SO, FO and IO. Our version of DipBlue works exactly the same as the original DipBlue described in [11].

DipBlue's advisers are part of the SO, and are used to calculate the utility value for possible moves in the same way as originally. Unlike with AlphaDip, in DipBlue each move is a single order for a unit, and each order has a utility value assigned by the advisers. After finding the best orders, the SO suggests them to the PR, who asks the FO to negotiate any deals it thinks are necessary. The FO implements DipBlue's negotiation strategy, requesting supports from its allies for any moves that could use them and negotiating alliances and peace deals.

The execution of DipBlue's moves is not dependent on the success of negotiations with other players, though any supports may increase the likeliness of those moves. As a result, the PR will always execute moves suggested by the SO, regardless of the result of any negotiations the FO attempts.

The IO updates opponent trust values in the same way the original DipBlue does. Unlike AlphaDip, DipBlue's IO does not predict opponent goals.

5.3 AlphaWolf

Werewolves of Miller's Hollow was also chosen to draw an implementation of the Alpha architecture. As far as we know, no frameworks are available to develop agents for this game. We have implemented a game server using the Jade multi-agent framework[3], for which we implemented an agent to play the game – AlphaWolf.

In order to simplify the implementation, and because certain roles are more suited to be played physically with humans, we use a simplified version of the game with a subset of the original player roles and abilities. In our version of the

[3] http://jade.tilab.com

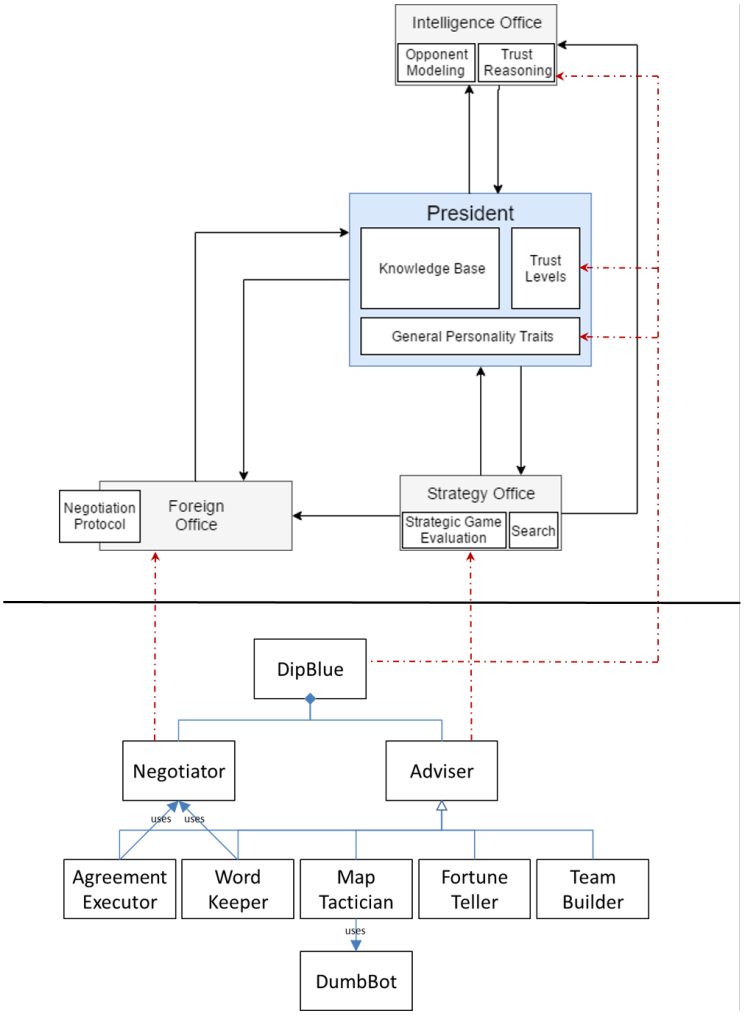

Fig. 4. Mapping DipBlue to the Alpha architecture.

game, there are 4 possible roles for the players: werewolves, villagers, seers and doctors. Werewolves have the goal of killing every other non-werewolf player in the game while every other player has the goal of killing the werewolves. The werewolves, seers and doctors each have a special ability that they can secretly perform during the night phase of the game. Werewolves can collectively vote on an enemy player to kill during the night. The seers can choose any player to investigate during the night, learning its secret role. Finally, doctors are able to choose a player, who if attacked by the werewolves during the night will be healed and remain in the game, informing the doctor that this happened.

The President. AlphaWolf's PR works as described in Section 4. At the start of each phase of the game it requests the IO to update opponent trust values and predicted goals. In the context of this game, a player's goal is tied to its role and as such predicting a player's goal means predicting the likeliness that a player has a certain role. In the PR's knowledge-base, this means that a probability is associated with each player-role pair. Role probabilities for a given player add up to 100%, so that if a certain role has a 100% probability, the PR knows that player's role and, consequently, its goal. A player's trust is represented by a positive or null real number, where 1 means neutral trustworthiness, 0 means no trustworthiness and values above 1 mean progressively higher trustworthiness.

The PR requests the SO to suggest a good move. The notion of move depends on the current phase of the game, but it always involves choosing a player to either vote out of the game or as a target for the player's ability during the night phase. After a player is suggested by the SO, and depending on whether the current phase of the game allows negotiation between the players, the PR may ask the FO to attempt to negotiate with the other players for joint votes against some player, or requests for investigation or healing. When the FO has finished negotiating, the PR decides to either vote or target a certain player for its special action. If no deal has been reached, the player is randomly chosen from the list of players suggested by the SO, with players with higher utility being more likely to be picked; otherwise, if a deal has been reached for a specific player, the PR will take the action it agreed to on the deal.

The Strategy Office. AlphaWolf's SO implements a simple strategy to suggest potential players to either attempt to eliminate or protect, depending on the current phase of the game and the player's role and goals. This is done by assigning each player a threat score, which is a measure of what roles and goals the player believes an opponent has (as calculated by the IO) and how dangerous these roles are to the player. In general, if a player is on the werewolf faction, roles that have the ability to gather more information or use abilities that hinder werewolves actions will have a higher threat level to the player. In the same way, if a player is on the villager faction, roles that have the ability to gather more information or hinder the werewolves are less likely to kill the player, and thus are less threatening. Depending on the current phase of the game and whether the actions available to the player will hinder an opponent (such as voting to kill it) or help another player (such as healing it), either this threat score is used as the utility for the move or its inverse is used, respectively. Equation 2 shows how the threat value T_p of player p is calculated, where n is the number of different possible roles a player can have, B_i is the base threat value for role i and C_i is the current estimated likelihood that player p has role i.

$$T_p = \frac{\sum_{i=1}^{n} B_i \times C_i}{\sum_{i=1}^{n} B_i} \tag{2}$$

The SO also has the purpose of calculating the utility of deal proposals. This utility is based on the previously mentioned threat value of the proposer of the deal, the threat value of the player whom the deal concerns, what type of action the deal is proposing and the trust of the player in the proposer of the deal. This calculation is described in Algorithm 3. The type of action proposed and the threat values for the proposer and the player affected by the proposal are used to calculate two values – one for the proposer and one for the target of the proposed action – representing how much the player is willing to help the proposer and hurt the target. These two values are then multiplied together with the trust on the proposer, representing how much the player trusts the proposer to abide by the deal and not take any actions against him, to reach the final utility value for the deal (line 9).

Algorithm 3 AlphaWolf SO Deal Utility Calculation

```
1: target ← getTargetFromDeal()
2: proposer ← getProposerFromDeal()
3: proposerValue ← 1 / getThreatValue(proposer)
4: if dealActionIsPositive() then
5:     targetValue ← 1 / getThreatValue(target)
6: else
7:     targetValue ← getThreatValue(target)
8: end if
9: return  targetVales × proposerValue × getTrustFromPR(proposer)
```

The Foreign Office. In Werewolves of Miller's Hollow, players can only communicate during certain phases of the game, namely the discussion phase and, for werewolves, the night phase. As the game is very reliant on communication between players, negotiation is very important in order to obtain effective players. Otherwise, players would not be able to coordinate their votes or use their abilities during the night. This is the purpose of the FO.

AlphaWolf's FO implements a simple negotiation strategy during the day phase, where each player proposes a joint vote against another player who they think is the most threatening, as well as other agreements such as investigation or heal requests, depending on their levels of trust with other players. Players then wait for agreement confirmations from their opponents, locking the agreement in place if they receive a confirmation. In each negotiation round, players compare the utility of the proposals they receive with their concession value, which is based on their own proposal and decreases over time, and decide either to continue waiting or accept another proposal, retracting their own.

In the case of the nightly negotiation phase for werewolves (where they coordinate to choose a victim), the strategy employed by the FO can be even simpler, since the werewolves have complete information about who the other werewolves are and can thus assume that they are working towards the same

goals. In this case, consensus is reached by means of a sealed bid mechanism, where each werewolf proposes one player to vote for as well as their preference level for that player. After all werewolves have made their proposals, the bids are counted and the player with the highest preference level among all werewolves is selected, with every werewolf voting for it.

The Intelligence Office. AlphaWolf's IO has the function of attempting to predict a player's goals and its trustworthiness. As mentioned previously, since a player's goals are tied to its role in the game, predicting its goals is a matter of predicting its role. In order to predict an opponent's role, the IO analyses the proposals and votes of that player over the course of the game. The predicted role probabilities for that opponent are thus a function of the threat values of the players that opponent voted against (or proposed to vote against), and the rounds in which that player took those actions. Algorithm 4 describes the calculation for the prediction of opponent goals by the IO.

Algorithm 4 AlphaWolf IO Role Prediction Calculation

```
1:  pastRounds ← getPastRoundsFromPR()
2:  opponents ← getOpponentsFromPR()
3:  opponentGoals ← getOpponentGoalsFromPR()
4:  for all op ∈ opponents do
5:      for all round ∈ pastRounds do
6:          roundAgeFactor ← getAgeFactor(round)
7:          for all actions ∈ getOpponentActions(op, round) do
8:              target ← getTarget(action)
9:              voteDamage ← getThreatValue(target) / getAverageThreatValue() × roundAgeFactor
10:             scalingFactors ← calculateRoleScalingFactors(voteDamage)
11:             for all r ∈ opponentGoals[op] do
12:                 r ← r× scalingFactors[r]
13:             end for
14:             normalizeOpponentGoals(opponentGoals[op])
15:         end for
16:     end for
17: end for
18: return opponentGoals
```

The IO searches through each player's past actions and for each vote or proposal that player made it calculates a vote or proposal damage value (line 9 in Algorithm 4). This value indicates the likelihood that an action was taken with the intent of damaging the player's faction and is based on the threat values of the players who are the targets of that opponent's actions. A high threat value for the target of the action indicates that the action was not very damaging to AlphaWolf's faction, and even may have been helpful, and a low threat value indicates a damaging action, as that opponent was voting against players that are considered likely to be allies.

For each action, its vote damage is then used to calculate a scaling factor for each possible opponent role (line 10), which is finally used to scale the role probabilities of each role proportionally (lines 11-12). Each role has a different scaling factor because certain roles are more likely to have more information than others: seers have a higher scaling factor than doctors, and doctors have a higher scaling factor than villagers. In this way, if an opponent takes a damaging action, the likelihood that it is a seer comparatively diminishes more than the likelihood that it is a villager, since a seer would be less likely to commit damaging actions against the villagers, having more information about the player roles. These scaling factors are then multiplied with each current role probability, and the values for the roles are then re-scaled back so that they total 100% (line 14).

To calculate trust values, the IO analyses the previous round and checks for each opponent if it kept any agreements it accepted or if it voted against the player. If an opponent broke an agreement or voted against the player, its trust value is decreased by a certain amount, otherwise its trust value increases.

6 Experimental Validation and Evaluation

In order to validate our implementation of the Alpha-based agents described in Section 5, we have setup a number of experimental scenarios. Besides illustrating the correct functioning of the agents themselves, these scenarios also helped in evaluating individual architecture components and, when available, enabled us to compare the performance of the agents with other state-of-the-art players.

6.1 AlphaDip Validation Setup

We compared AlphaDip with three previously developed Diplomacy playing agents: DumbBot, DipBlue and D-Brane. For that, we performed tests similar to those reported in [10] and [6]. It should be noted that unlike the D-Brane tests reported in [6], which assumed D-Branes always formed a coalition against every other agent in the game, we allow our agents to negotiate at will, establishing and breaking coalitions.

In each experimental setup, we tested AlphaDip using 3 distinct configurations, in order to separately assess the impact of negotiation and opponent/trust modeling in its performance: (i) using only the PR and SO, (ii) including the FO, and (iii) using all four PR, SO, FO and IO.

For every configuration, in each experiment we played a number of games of Diplomacy, stopping after 40 game phases. After the end of a game, we ordered players by ranking, from 1st to 7th, and collected ranking results. Ranking is determined by the number of supply centers a player controls at the end of the game (in case it is still alive), or by the game phase in which the player was eliminated. Players with more supply centers or eliminated at later phases have a higher rank in the game and are thus considered better. For configurations without the FO (and thus, with no negotiation capabilities) we played a total of 100 games. For configurations including the FO we set the negotiation deadline

Table 2. Average rank of 2 AlphaDip when playing with 5 DumbBot.

2xAlphaDip & 5xDumbBot	
AlphaDip Config.	Avg. Rank
PR + SO	2.35 ± 0.15
PR + SO + FO	2.21 ± 0.21
PR + SO + FO + IO	2.11 ± 0.19

at 15 seconds per round. Since these tests take considerably longer to execute, we only played a total of 50 games per configuration.

All tests and experiments were performed using a laptop computer with 8GB of RAM and an Intel Core i5-6440HQ mobile CPU clocked at 3.5GHz.

6.2 AlphaDip Results

Similar to experiments reported in Ferreira's [10] and de Jonge's [6] works, in each experiment we had two instances of AlphaDip playing against 5 instances of DumbBot. Combining the ranks of our AlphaDip agents, the best possible average rank is 1.5, while the worst possible average rank is 6.5. We can compare the average ranks of AlphaDip with the average ranks of DipBlue and D-Brane reported in [10] and [6], respectively. The best rank achieved by DipBlue in the tests performed by Ferreira is 3.57 [10], while the best rank obtained by D-Brane in the tests performed by de Jonge is 2.35 [6].

Table 2 shows the average rank obtained by AlphaDip in each configuration (standard deviation is also included). These results show that AlphaDip plays significantly better than the DumbBot and DipBlue, even without negotiation, opponent and trust modeling capabilities. However it also appears that the inclusion of IO and FO only has a small effect on the performance of the agent. A t-test performed on these results obtains a score of 0.554 when comparing the second with the third configuration, and a value of 0.109 when comparing the first with the third. This points towards statistically significant performance differences between the base configuration (PR+SO) and the full one (PR+SO+FO+IO), although further testing is required to confirm this hypothesis.

In order to test how well AlphaDip performs against other, more advanced, agents we also performed an experiment including two AlphaDips and two D-Branes (in this case without negotiation capabilities[4]), together with 3 DumbBots. The results of these tests are shown in Table 3. Similarly to the previous experiment, these tests show that AlphaDip outperforms D-Brane, especially when it can use the negotiation and opponent modeling capabilities provided by the FO and IO. When playing only with the SO, AlphaDip performs only slightly better than D-Brane. Because AlphaDip's SO is based on D-Brane's strategic module, however, this difference is not statistically significant.

[4] The public versions of D-Brane do not include negotiation capabilities (`http://www.iiia.csic.es/~davedejonge/bandana`).

Table 3. Average ranks in games with 2 AlphaDip, 2 D-Brane and 3 DumbBot.

2xAlphaDip, 2xD-Brane & 3xDumbBot		
AlphaDip Config.	AlphaDips Avg. Rank	D-Branes Avg. Rank
PR + SO	2.84 ± 0.17	2.91 ± 0.16
PR + SO + FO	2.37 ± 0.21	3.19 ± 0.24
PR + SO + FO + IO	2.29 ± 0.20	3.25 ± 0.24

Table 4. Average ranks in games with 2 AlphaDip, 2 DipBlue and 3 DumbBot.

2xAlphaDip, 2xDipBlue & 3xDumbBot		
AlphaDip Config.	AlphaDips Avg. Rank	DipBlues Avg. Rank
PR + SO	2.38 ± 0.16	5.03 ± 0.20
PR + SO + FO	3.56 ± 0.26	3.44 ± 0.32
PR + SO + FO + IO	2.3 ± 0.22	4.73 ± 0.29

In order to complement the previous experiments, we also tested AlphaDip in an environment with a higher number of negotiating agents: 2 AlphaDips play with 2 DipBlues and 3 DumbBots. The 2 DipBlues played with the standard adviser configuration described in [10]. Results are shown in Table 4.

These results show that when AlphaDips are playing with a PR+SO configuration or with all modules running they get a similar average rank, between 2.3 and 2.4. A statistical t-test gives us a value of approximately 0.653, which shows that the difference observed is not statistically significant. The inclusion of negotiation and trust reasoning does not seem to significantly affect the performance of AlphaDip in this experiment. On the down side, negotiation without opponent modeling seems to be counterproductive, as using the FO without the IO (meaning that AlphaDip negotiates without making any attempt to predict opponent goals or trustworthiness) brought the worst results in this experimental setup – the average rank decreases to 3.56.

This lack of impact from negotiation resembles results obtained by de Jonge in [6], where he points out that even though the NB^3 algorithm manages to find good joint moves (when they exist), their impact in the overall result of the game is negligible. Negotiating joint moves for only the current round is not enough to significantly increase the performance of the players – in order to obtain better results one would have to attempt to negotiate further rounds ahead as well.

Another interesting observation is that, while AlphaDips do obtain a small increase in their average rank when all modules are running as compared to having just PR+SO, DipBlues themselves also benefit from this setting: DipBlues obtain a better average ranking of 4.73 when playing against AlphaDips with all modules active, compared with an average ranking of 5.03 when playing with AlphaDips incapable of negotiation. These observations were also corroborated by de Jonge's observations in [6], where he found that other agents could also benefit from the deals discovered by agents running the NB^3 search algorithm.

Table 5. Win percentages and average number of remaining villagers for a team of 8 villagers playing against 2 werewolves.

Villagers Config.	Win %	Avg. # Villagers
PR + SO	27%	3.56
PR + SO + FO	46%	3.20
PR + SO + FO + IO	73%	4.88

6.3 AlphaWolf Validation Setup

Testing AlphaWolf is not as straightforward as testing AlphaDip, due to the lack of agents developed for this game. Nevertheless, we conducted experiments that allowed us to test the relative performance of our agent with different configurations of active modules.

In order to test the impact of each module in the performance of the agent, we opted to have AlphaWolf werewolf agents always playing with all modules running, and changed only the configurations of AlphaWolf villager agents. This way, we can easily see the effect each module has on the performance of villagers. As before, We tested 3 different configurations for this agent: (i) using the PR and the SO, (ii) including the FO, and (iii) using all four PR, SO, FO and IO.

In each test we had the agents play 100 games in a 10 player game where 2 of the players were werewolves, and the remaining 8 were from the villager faction, with 1 seer, 1 doctor and the remaining 6 being standard villagers. This ratio of werewolf to villager players was chosen because it is the recommended ratio in the official Werewolves of Miller's Hollow rules [8]. We recorded the win percentage for the villagers over those games as well as the average number of villager agents left alive at the end of the game when the villager faction won.

6.4 AlphaWolf Results

Unlike with Diplomacy, results for the AlphaWolf agents show that negotiation, trust and opponent modeling are effective features, with a significant impact on agent performance (see Table 5). With the inclusion of the FO and the IO, performance steadily increases from a 27% to a 46% win ratio, and finally to a 73% win ratio with all modules active. The inclusion of the IO also significantly increases the number of villagers remaining alive in games where this faction wins. This indicates that, with the inclusion of trust and opponent modeling, players are able to identify werewolves much earlier in the game, allowing for quicker victories.

One possible explanation for the difference in the relative impact of negotiation, trust and opponent modeling in the game of Werewolves of Miller's Hollow as compared with Diplomacy is that in the latter agents already implicitly have an idea of their opponents' goals. Since in Diplomacy the rules of the game encourage players to capture supply centers, one can assume that other players will try to maximize their number of supply centers over the course of the game. On the other hand, in Werewolves of Miller's Hollow players have no way to

know, at the initial stage of the game, what their opponents will be trying to do. This means that players have no way to predict an opponent's utility function at the start of the game, and must analyze its actions in order to predict the opponent's goal.

Negotiation may also have a greater impact in Werewolves of Miller's Hollow because each player only has a single vote to affect the round. Without coordination and organizing joint votes, players have a hard time completing their goals. In Diplomacy, players can have differing numbers of supply centers and units, which allows certain players to affect the outcome of the rounds more than others; this means that strong players can use their superior strength to obtain their objectives even without negotiation.

7 Discussion

While there have been several attempts to develop general game playing AIs, the task has not proven to be easy, and the available proposals often come with several important limitations. The path that has been taken throughout time is to address successively more and more complex characteristics of the environment faced by agents: from deterministic, sequential and perfect information games, to stochastic and imperfect information games. The next frontier may well be games demanding for social relationships, where negotiation and trust modeling are essential components to establish cooperation, a need that is magnified in the presence of simultaneous moves.

The low coverage of games that include this social element is one of the biggest limitations of current approaches such as Zillions of Games [4] and the General Game Playing project [19], which focus mostly on abstract board games involving no or little cooperation. The Alpha architecture seeks to bridge the gap between the current state-of-the-art and cooperative negotiation games.

However, it is important to note that the Alpha architecture and framework do not constitute in themselves a general game playing AI, as they do not offer some essential capabilities that other general game playing AIs possess. For example, a way for the developed agents to abstractly understand the rules of the different games is entirely missing (see [2] for a glimpse of the challenge). In order to obtain a truly general game playing AI it is necessary for the developed agents to be able to learn different rule sets for different games that the developers might not even themselves know, for example by reading a description language file (using formats such as [21] or [5]) that describes the rules for these games.

The proposed architecture is instead intended to serve as a strong framework, upon which game-specific agents can be implemented, that identifies, organizes and generalizes the use of a number of different capabilities required to do well at negotiation games. By using the proposed architecture, developers can easily implement agents that make use of a combination of search strategies with negotiation, opponent modeling and trust reasoning capabilities, in a general and standardized way. This basis can be used in the future to develop a true general game playing AI that is able to play a variety of cooperative negotiation games.

This can be done by adding some improvements over the current work, while keeping the same role for each of the modules of the Alpha architecture.

8 Conclusions and Future Work

The objectives of this work were the study of what elements were necessary to create effective agents that could play cooperative negotiation games, and to develop a generic architecture including these elements, which could be used to facilitate the development of effective agents for a wide variety of games.

We tested the proposed architecture by developing agents for two very different cooperative negotiation games, and believe that the proposed Alpha architecture is generic enough to be applied to many other different games. The two most relevant agents developed using the Alpha architecture and framework were AlphaDip and AlphaWolf. AlphaDip is an agent with strategies inspired by D-Brane and DipBlue, with the inclusion of opponent modeling to make predictions about an opponent's intention to capture certain supply centers, as well as a negotiation strategy for the establishment of coalitions, which was not present in D-Brane. AlphaWolf is a Werewolves of Miller's Hollow agent that also includes negotiation, trust and opponent modeling capabilities, allowing it to predict its opponents' roles and negotiate deals accordingly.

The results of the tests performed using these two agents show that AlphaDip was in general superior to both DipBlue and D-Brane, obtaining better average ranks in the games played. However, the inclusion of negotiation, trust reasoning and opponent modeling capabilities did not have a very large impact on the performance of the agent. On the other hand, results obtained for AlphaWolf show that the inclusion of the Foreign Office and Intelligence Office had a larger impact in the performance of the agent. This indicates that negotiation, trust and opponent modeling are more important in Werewolves of Miller's Hollow than in the Diplomacy game.

We believe that these results are positive and the inclusion of negotiation, trust reasoning and opponent modeling capabilities generally improved the performance of the agents, though the impact was much greater for AlphaWolf than for AlphaDip. We also believe that the developed architecture and framework are a helpful contribution to the field by facilitating the development of agents with these capabilities. However, while the developed architecture is very modular and allows agents to be built upon it and make use of negotiation, trust and opponent modeling, there is lots of room for improvement. The Alpha architecture allows developers to define different negotiation strategies, trust reasoning and opponent modeling approaches, which may be tailored to a specific game. However, this process can be made simpler with the inclusion of generic strategies (such as D-Brane's NB^3 algorithm) that can be applied equally to any environment. The inclusion of a generic way to predict opponent goals and strategies, calculate trust values and decide what deals to accept, based on the knowledge-base of the President, would further simplify the process of developing an efficient agent.

The agents implemented during the course of this work, while generally efficient, could also be improved. One major improvement to AlphaDip is to allow the agent to search for and negotiate movement commitments for several rounds ahead, instead of only the current round. As for AlphaWolf and the implemented Werewolves of Miller's Hollow server, one key improvement would be the capability for AlphaWolf to use strategies involving bluffing, by for example making opponents believe it has a different role than its true role, a strategy human players frequently use. If correctly implemented, this ability could make AlphaWolf much more effective, especially when playing with human opponents.

References

1. Einar Nour Aouni and Leif Julian Øvrelid. *Negotiation for Strategic Video Games*. Master thesis, Norwegian University of Science and Technology, 2013.
2. Édouard Bonnet and Abdallah Saffidine. *On the Complexity of General Game Playing*, pages 90–104. Springer International Publishing, Cham, 2014.
3. Allan B. Calhamer. *The Rules of Diplomacy*. Avalon Hill, 4th edition, 2000.
4. Zillions Development Corporation. Zillions of Games. `http://www.zillions-of-games.com`, 2016. Accessed: 2017-05-16.
5. João Correia. *PGDL: Sistema para definição genérica de jogos de Poker*. Master thesis, Universidade do Porto, 2013.
6. Dave de Jonge. *Negotiations over Large Agreement Spaces*. PhD thesis, Universitat Autonòma de Barcelona, 2015.
7. Dave de Jonge and Carles Sierra. Negotiation Based Branch & Bound and the Negotiating Salesmen Problem. In *Proceedings of the 14th International Conference of the Catalan Association for Artificial Intelligence*, Lleida, Catalonia, Spain, 2011.
8. Philippe des Pallières and Herv Marly. Werewolves of Miller's Hollow: The Village. Lui-même, 2009.
9. Alexis Drogoul. When ants play chess (Or can strategies emerge from tactical behaviours?). In Cristiano Castelfranchi and Jean-Pierre Müller, editors, *From Reaction to Cognition: 5th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW '93 Neuchâtel, Switzerland, August 25-27, 1993 Selected Papers*, pages 11–27. Springer, 1995.
10. André Ferreira. *DipBlue: a diplomacy agent with strategic and trust reasoning*. Master thesis, Universidade do Porto, 2014.
11. André Ferreira, Henrique Lopes Cardoso, and Luís Paulo Reis. Strategic Negotiation and Trust in Diplomacy – The DipBlue Approach. In Ngoc Thanh Nguyen, Ryszard Kowalczyk, Béatrice Duval, Jaap van den Herik, Stephane Loiseau, and Joaquim Filipe, editors, *Transactions on Computational Collective Intelligence XX*, pages 179–200. Springer International Publishing, Cham, 2015.
12. Michael Genesereth, Nathaniel Love, and Barney Pell. General Game Playing: Overview of the AAAI Competition. *AI Magazine*, 26(2):62–72, 2005.
13. Sarit Kraus, Ramat Gan, and Daniel Lehmann. Designing and Building a Negotiating Automated Agent. *Computational Intelligence*, 11(972):132–171, 1995.
14. João Marinheiro and Henrique Lopes Cardoso. A Generic Agent Architecture for Cooperative Multi-agent Games. In *Proceedings of the 9th International Conference on Agents and Artificial Intelligence - Volume 1: ICAART,*, pages 107–118, 2017.

15. John Nash. Non-Cooperative Games. *The Annals of Mathematics*, 54(2):286–295, September 1951.
16. David Norman. David's Diplomacy AI Page. `http://www.ellought.demon.co.uk/dipai/`. Accessed: 2017-05-16.
17. University of Essex. The GVG-AI Competition. `http://www.gvgai.net`. Accessed: 2017-05-16.
18. Luís Paulo Reis, Pedro Mendes, Luís Filipe Teófilo, and Henrique Lopes Cardoso. *High-Level Language to Build Poker Agents*, pages 643–654. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
19. Sam Schreiber. GGP.org – General Game Playing. `http://www.ggp.org`. Accessed: 2017-05-16.
20. David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.
21. Michael Thielscher. The General Game Playing Description Language is universal. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI-11)*, pages 1107–1112, 2011.