Universiteit Leiden
Opleiding Informatica

Tree Search Methods

for Diplomacy Agents

Name: Xander Croes

Date: 05/07/2016

1st supervisor: Dr. W.A. Kosters
2nd supervisor: B.J.G. Ruijl MSc

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

After the defeat of the human Go champion by Artificial Intelligence (AI) methods, Diplomacy has been named among the choices for a next-to-beat game. Diplomacy is a deterministic boardgame, in which players attempt to take control over Europe. The game is considered hard because it has multiple players, simultaneous play and free communication.

Our goal is to make an AI agent for Diplomacy with a very basic strategy. We performed a preliminary study to find out if a program using tree search methods performs well in Diplomacy and whether or not it will be able to recognize good patterns. We made an agent using a basic form of Monte Carlo Tree Search (MCTS) and another using Minimax. These agents focus on the tactical decisions and ignore communications between players. Several MCTS agents, each with a different number of playouts, are tested against each other on small simplified boards.

It becomes immediately clear that Minimax is not feasible at all without pruning. This is because even for a tiny board and expanding to a limited depth the branching factor is too large. While MCTS can take considerable time to compute its moves as well, its decisions can be made well within the time limits included in the rules. Against random play MCTS managed to win most of the games, depending on the amount of playouts it performed. Furthermore, some of the strategic patterns adopted by human players were also found in the plays of the MCTS agent.

The MCTS agent should be further tested in the full game (with all possible moves and with communications) and against other agents, however it is recommended to add the Upper Confidence Bound heuristic to the MCTS algorithm to improve its efficiency.

Contents

1 Introduction

Artificial Intelligence (AI) research has been around since the 1950's. In games, one of the challenges is to beat human players. Often it takes many years to find an appropriate strategy, while AI methods are becoming increasingly elaborate. For this master's thesis at Leiden University, supervised by Walter Kosters and Ben Ruijl, the aim is to make a simple agent for Diplomacy. Diplomacy is a social boardgame where players try to conquer 20th century Europe.

Tree Search methods have been particularly strong in board games [1, 2]. The big drawback is that it takes a lot of time and memory to build a tree of the complete search space. Therefore our method of choice is a simplified Monte Carlo Tree Search (MCTS). We can, to some degree, decide for how long this method runs, and only the visited nodes are expanded. Human champions of many games have been beaten by AI methods using MCTS as (part of) their strategy. Two famous examples are the defeat of Garry Kasparov in 1997 by IBM's Deep Blue [1] (chess) and, in 2016, the defeat of Lee Sedol by Google's AlphaGo [2] (Go).

We are interested to see the performance of our methods, both in terms of winrate and computation time. Furthermore, we are interested to see whether or not MCTS is able to recognize certain strategic patterns adopted by the players.

The paper overview is as follows. Section 2 contains the rules for Diplomacy. Section 3 contains related work. In Section 4 we describe MCTS. In Section 5 we describe minimax. Section 6 describes our agents. In Section 7 we discuss our experiments and results. Finally, Section 8 contains our conclusions and future work.

2 Diplomacy

Diplomacy is a strategic (almost fully) deterministic boardgame with simultaneous actions [4]. The goal is to capture over half of the control points (Section 2.1) in order to win. Players can work together in trying to achieve this goal, or they can betray one another, anything is allowed (including cheating if it goes undetected!). In addition to the players, Diplomacy requires one referee, also called the *adjudicator*. The referee controls the gameflow and enforces the rules. The only non-deterministic part of the game, assigning starting positions (one of the empires), goes by the roll of dice.

The original game is about the politics and armed conflicts taking place in Europe during the early twentieth century. It can be played by up to seven players, where each player takes control of one of the great powers of that time. Figure 1 shows the standard Diplomacy board, including starting units.

We made our own Diplomacy server (including adjudicator) in Java, and implemented the rules (with the exception of convoys), according to Kruiswijk's guidelines on how to build an adjudicator [5]. However, not all testcases have been run for the adjudicator, so there might still be some errors. In the remainder of this section we provide a brief description of the game rules. The rules, including detailed examples, can be found at [4].

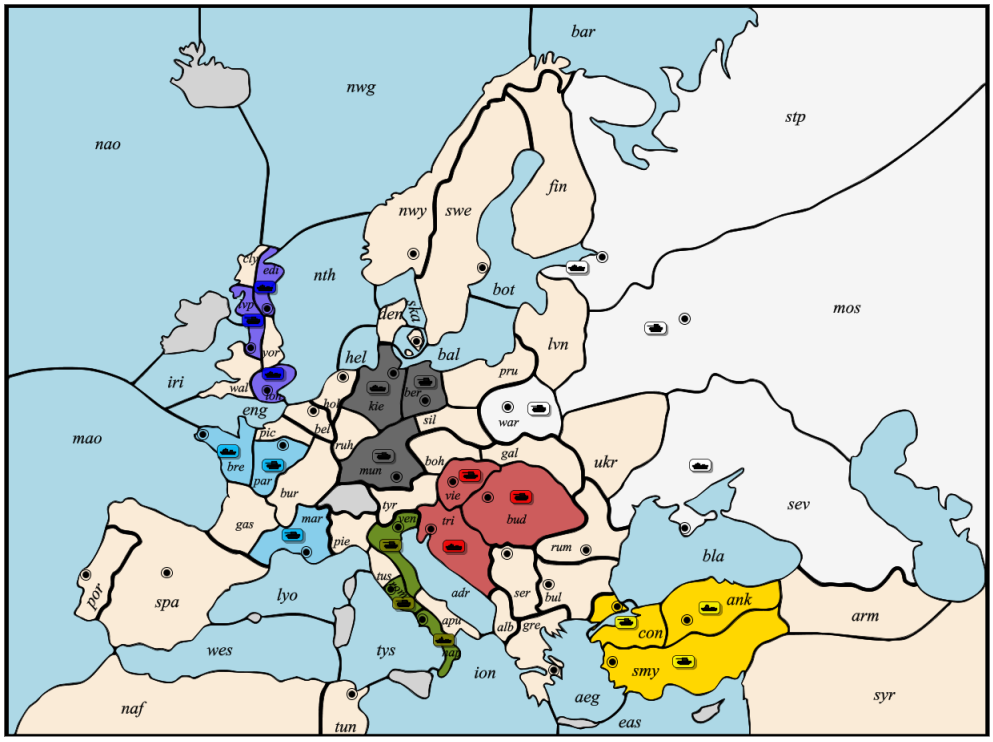

Figure 1: The standard Diplomacy board. Source: [3].

2.1 Board

The board consists of the playing pieces, called units, and a map. The map is divided in positions called provinces. A province has one of the following terrain types: water, coast or land. Furthermore, a province can contain a supply center. These supply centers can be captured by players by having a unit occupy it at the end of a year. When captured by a player, the supply center provides a control point to the capturing player. When a player loses a supply center, he also loses the control point it provided. At the start of a game, some of the supply centers will already be controlled by players. These supply centers are called home supply centers.

A player may only have one unit for each control point he or she has. There are two types of units: armies and fleets. Fleets can occupy water and coastal provinces, while armies can occupy coastal and inland provinces. Futhermore, each province can only be occupied by one unit at a time.

2.2 Orders

The actions in Diplomacy are called orders. The different orders are: hold, move, support, convoy, retreat, disband and build. Each unit can only perform one order per turn. Issuing multiple orders to a single unit results in the last valid order being used. Figure 2 shows which order can be issued in which phase.

	Hold	Move	Support	Convoy	Retreat	Disband	Build
Diplomatic	✖	✖	✖	✖	✖	✖	✖
Order	✔	✔	✔	✔	✖	✖	✖
Adjudication	✖	✖	✖	✖	✖	✖	✖
Retreat	✖	✖	✖	✖	✔	✔	✖
Adjustments	✖	✖	✖	✖	✖	✔	✔

Figure 2: The possible orders for each phase.

Hold Hold position. Also the default order for a unit if its order was invalid or it did not receive an order.

Move Move a unit to an adjacent province. Units can only move to provinces of which the type matches to the unit's type.

Support Increase the strength of another unit. The supporting unit must be able to reach the supported unit's destination, but does not move.

Convoy Ferry an army over bodies of water. Convoy orders can only be issued to fleets. The destination province has to be either directly reachable by being an adjacent province, or indirectly in a chain of convoys, where each convoying unit is in a province adjacent to the previous/next convoying unit. The convoying unit does not move.

Retreat Move a dislodged unit to an adjacent province.

Disband Remove a unit from play if it is dislodged or if the player controls too many units.

Build Add a new unit to the game.

For the full requirements for validity of orders the reader is referred to the manual [4].

2.3 Turns

Diplomacy uses simultaneous turns, meaning that all players issue their orders at the same time. Diplomacy uses a seasonal system with two different turns: Spring and Fall. A spring turn and a fall turn together form a year. As shown in Figure 3, the spring turn consists of the Diplomatic phase, the Order phase, the Adjudication phase and the Retreat; and the fall turn has the same phases plus an additional phase, called Adjustments phase.

2.4 Adjudication

During adjudication the referee first makes sure that all orders are valid. After that, the strength of the units is used to determine which orders fail and which pass. The assisting orders (support, convoy[1]) are evaluated first:

[1]Convoys are not implemented in our program

	Spring	Fall	Description
Diplomatic	✔	✔	Players engage in diplomatic negotations with each other
Order	✔	✔	Players prepare their orders and send those to the referee
Adjudication	✔	✔	The referee decides the outcome of the presented orders
Retreat	✔	✔	Players prepare orders for their dislodged units, referee decides outcome
Adjustments	✖	✔	Players give build/disband orders to match their control points, referee decides outcome

Figure 3: The phases that each turn contains.

- If a supporting unit is dislodged, the support is cut.
- If a supporting unit is attacked by a unit from a different player, from any province other than the province where support is given, the support is cut.
- If a convoying unit is dislodged, the convoy is disrupted.

If support is cut, the supported unit does not receive the increase in strength from the supporting unit. If a convoy is disrupted, and no other convoy is available for the convoyed unit, the convoyed unit's order fails. However, note that it is impossible for a player to dislodge his/her own units, and therefore a player cannot cut/disrupt his or her own supports and convoys. Finally the move orders are evaluated:

- If there are no other orders with the same destination, the order will pass.
- If multiple orders have the same destination, but one order has a higher strength than all the others, the order with the highest strength will pass, other move orders with that destination will fail and if a unit was holding at the destination province and its strength was not the highest, this unit will be dislodged.
- If multiple orders have the same destination and some share highest strength, there will be a standoff.
- If the unit tries to swap provinces with another unit, there will be a head-to-head battle.

When a standoff occurs, all the move orders with the destination of the standoff will fail. Furthermore, the province where the standoff occurred will be inaccesible for retreat this turn. When a head-to-head battle occurs, if both units have the same strength or are both from the same player, both orders will fail. Otherwise, the unit with the highest strength wins and the other unit is dislodged.

3 Related work

In 2002 the Diplomacy AI Development Environment (DAIDE) [6] was created, to aid in the development and testing of agents for Diplomacy. DAIDE consists of only a communications framework, although the adjudicator and several basic agents are often considered part of DAIDE too. In 2009, extending on the ideas of DAIDE, Fabregues and Sierra created dipGame [7], a testbed for Diplomacy agents. In addition to the communications protocol, dipGame also provides a gamemanager and a framework for making agents. The Diplomacy server can be chosen by the user, but dipGame comes with a default configuration for Parlance.

3.1 Israeli Diplomat

One of the first agents for Diplomacy was the Israeli Diplomat [8]. Kraus et al. proposed a multi-agent system (MAS), where each agent takes on a role, such as *Prime Minister*, *Ministry of Defense*, *Foreign Office*, *Military Headquarters*, *Intelligence* and *Strategies Finder*. The *Prime Minister* has personality traits and keeps track of rules, alliances and the state of the game. The *Foreign Office* is responsible for maintaining relationships with other players and *Intelligence* tries to estimate what relations other players have amongst each other. The *Ministry of Defense*, *Military Headquarters* and *Strategies Finder* are responsible for making orders, taking into account the relations between players and the *Prime Minister*'s personality and information.

3.2 Bordeaux Diplomat

Clearly, the Israeli Diplomat has been an inspiration for other agents, such as the Bordeaux Diplomat [9], which is also a MAS. However, whereas the focus of the Israeli Diplomat is more on communications, the focus of the Bordeaux Diplomat lies more on strategy. The Bordeaux Diplomat is composed of a negotiator and a strategic core. The strategic core has knowledge of the rules and game state, but not about which power is being played. It can receive queries to evaluate the strategic importance of provinces and then uses a best-first search algorithm to propose strategies. The negotiator is responsible for keeping track of, and managing relations with other players. Furthermore it can query the strategic core for a set of strategies and then choose which strategy to follow.

3.3 LA Diplomat

The LA Diplomat [10] is a self-learning agent that uses pattern-weights. Shapiro et al. describe a pattern as a representation of a previous experience during gameplay, where the patterns represent partial positions on the board. They translated these positions to abstract graphs, only showing what moves are possible (omitting positional data). Each pattern is then assigned a weight, which is used as evaluation value of the actual state. With each play these weights are then updated through a self-learning algorithm.

3.4 DumbBot

DumbBot [11] is an agent with a simple strategy. It first calculates values for provinces and then moves to the provinces with the highest values, with a random chance to move to a province with a lower value. During calculation it takes into account the reachability for all units, location and neighbourhood to supply centers and who occupies those supply centers. DumbBot does not use negotiations, however, it has been extended in BlabBot to do so.

3.5 HaAI

HaAI [12] is another MAS, which creates an agent for all of its own units. Each agent can evaluate its surroundings to pick a best move individually, or the agents can be given a list of goals, allowing the agents to work together. One big drawback of this agent is that it cannot use convoys.

3.6 DarkBlade

The makers of DarkBlade [13] tried to combine MAS, personality traits and province values in a single strategy. Similar to HaAI, there is an agent for each of its own units, called a *General*. The generals propose orders for their units. In addition to the generals, there is an agent called *President*, which selects the best combination of orders based on the province values. All agents make use of the same set of personality traits.

3.7 Albert

Albert [14] is an agent which has been in development for 8 years. While the author calls it decent, version 4.0 (which is not the latest) achieved a winrate of 50% in a 1 human (several expert players) versus 6 Alberts challenge in 2009. For each power, Albert calculates the best moves and for each order iteratively calculates the probability that it will be played. Albert is also capable of a high level of communications.

4 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) [15, 16] is used in game theory on widely branching search spaces to find a good continuation. In MCTS we build a search tree by doing random sampling. The nodes of this tree contain the game state and counters for the amount of games that were won and played using the node. The branches represent the moves taken to reach the next game state. The tree is built in the following manner: first we select children according to some policy (for pure MCTS all children are selected equally often) until we reach a node with unvisited children; then we expand one of the unvisited children and do a random playout using this child; finally the *won* and *played* counters for all nodes in the selected path are updated. Usually MCTS consists of four steps:

Selection starting from the root, select child nodes until you reach a leaf node

Expansion add a child to the selected leaf node, unless the leaf node ends the game

Simulation play a game, doing random moves and continuing from the added child

Backpropagation update the nodes contained in the path from child to root with number of won/played games

Figure 4 shows a MCTS search tree. Here max/min to the left of the tree indicates whose turn it is. The squares below the nodes indicate who has won and the labels on the transitions from one node to another show which move was played. The numbers in the nodes show wins/plays. Notice that the first playouts for each node (dashed transitions to rewards) are not labelled with an action.

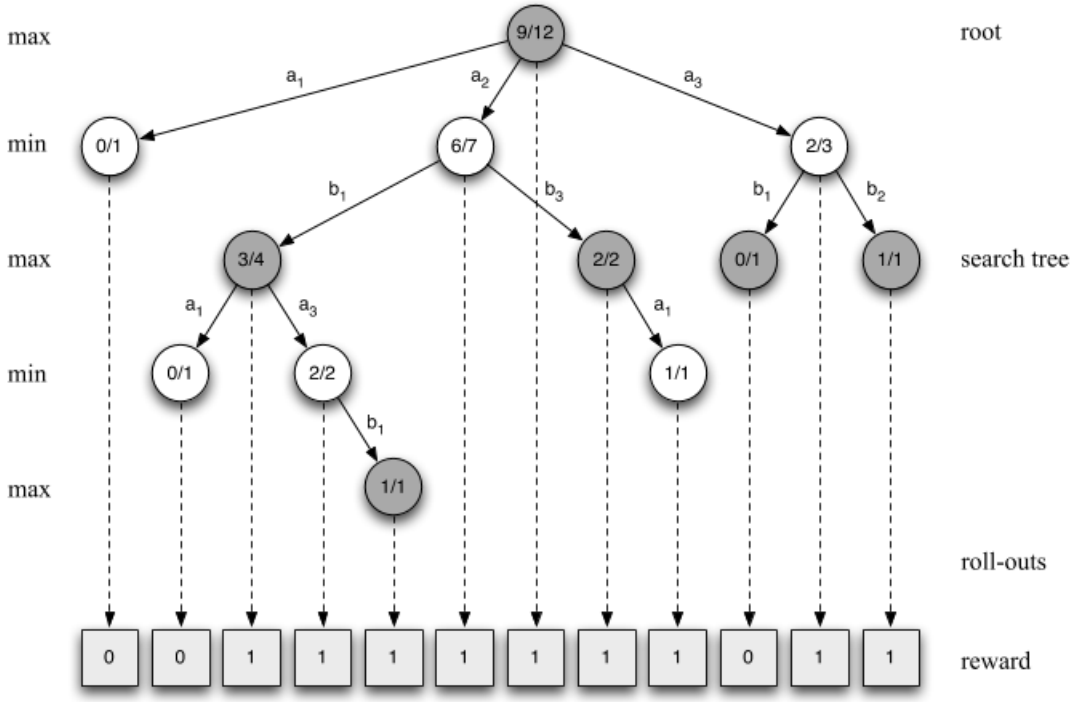

Figure 4: A Monte Carlo Tree Search in progress. Source: [19].

Brügmann's application of Monte Carlo methods on the game Go in 1992 [18] was one of the first times Monte Carlo methods were used on a board game, but not much was done with his work until around 2006, when Monte Carlo methods were first combined with Tree Search by Coulom [17]. In the same year, Kocsis and Szepesvári introduced the Upper Confidence Bound for trees (UCT) [19]. UCT controls the exploitation (higher values for nodes with high winrate) versus exploration (higher values for nodes with few plays) aspect for child node selection, where the child with the highest value is selected. UCT uses the

following expression to determine the value v_i for node i:

$$v_i = \frac{w_i}{n_i} + C\sqrt{\frac{\ln t}{n_i}}$$

Here:

- w_i is the number of wins using node i
- n_i is the number of games played using node i
- C is the exploration parameter (if $C = 0$, the exploration aspect is ignored and the node with the highest average win rate is selected)
- t is the total number of games played

The value v_i is used in the selection step of the MCTS algorithm, where the children with the highest values are chosen. The value for C is selected by the user and controls how exploration-driven the MCTS algorithm will be. A low C means w_i/n_i has a higher relative weight, thus increasing the affinity towards exploitation, whereas a high C increases the relative weight of $\sqrt{\ln t/n_i}$, increasing the affinity towards exploration.

5 Minimax and maximin

Minimax and maximin [15] are decision rules, originally intended for two-player zero-sum games, which are used to minimize the possible loss for a worst case scenario. We consider games where the players simultaneously make their moves. Generally the player tries to maximize his score, while his opponents try to minimize his score. Minimax yields the smallest value that a player can be forced to receive by his opponents if they do not know the player's actions, whereas maximin is the largest value the player can be sure to receive without knowing his opponents' actions. In minimax we assume that the player knows which moves his opponentswill take and can maximize after the minimization (player moves last), whereas in maximin we assume that the opponents know which moves the player will take and maximization comes before minimization (player moves first). To illustrate this consider Table 1 with scores for player A, where a_1 and a_2 are the moves player A can do and b_1 and b_2 are the moves for player B. Figure 5a shows the minimax search tree and Figure 5b shows the maximin search tree. Note that maximin corresponds to the paranoid situation where the other players know which move you will make, while minimax is optimistic. The maximin value is always smaller than or equal to the minimax value.
The minimax value $\overline{v_i}$ of node i is defined as:

$$\overline{v_i} = \min_{a_{-i} \subseteq A_{-i}} \max_{a_i \subseteq A_i} v_i(a_i, a_{-i})$$

and the maximin value $\underline{v_i}$ of node i is defined as:

$$\underline{v_i} = \max_{a_i \subseteq A_i} \min_{a_{-i} \subseteq A_{-i}} v_i(a_i, a_{-i})$$

	a_1	a_2
b_1	1	3
b_2	4	2

Table 1: Scores for player A.

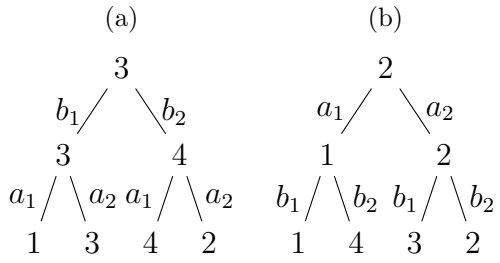

Figure 5: (a) Minimax and (b) Maximin Search Tree examples.

Here:

- A_i is the set of possible valid actions for player i.
- a_i is the set of actions taken by player i, for instance *French army Par holds & French fleet Bre moves to ENG* in Diplomacy.
- A_{-i} is the set of possible valid actions for the other players.
- a_{-i} is the set of actions taken by the other players.
- $v_i(a_i, a_{-i})$ is the value function for player i where player i takes action a_i, while the others take action a_{-i}

6 Agents

We implemented three different strategies: semi-random, basic MCTS and minimax. For now we ignore the diplomatic phase and agents do not support or convoy units other than their own.

6.1 Cartesian product

Considering the units $u_1, u_2, \ldots, u_k$ [2], we generate the sets $A_1, A_2, \ldots, A_k$, where A_i is the set of valid orders for unit u_i. In these sets, instead of true support/convoy orders (for instance *A Ruh S A Mun – Bur*), we have support/convoy orders which state which unit they are supporting/convoying like so: *A Ruh S A Mun*. We then generate the Cartesian product

[2]Note that it is possible that units are renumbered inbetween game phases.

$A_1 \times A_2 \times \ldots \times A_k$. Finally we expand the support/convoy order to true support/convoy orders and remove any tuples with convoy/support orders that are not valid. This means that tuples similar to *A Ruh S A Mun – Bur & A Mun – Kie* are not included in the final product. However, this is not a big issue, because a support order is meaningless when the order it supports is not given, and such a support order can be considered to be equivalent with a hold order.

6.2 (Semi-)Random

The random agent chooses a random tuple from the Cartesian product of its own units and delivers it to the adjudicator. The semi-random agent only differs from the random agent during the order phase of fall turns. During those phases the semi-random agent first tries to consolidate its positions, giving hold orders to units that are in a province that is a supply center which is not controlled by the agent. Then it generates the Cartesian product consisting of the hold orders and the valid orders of its other units. Finally, it chooses a random tuple from the Cartesian product and delivers the tuple to the adjudicator.

6.3 MCTS

The MCTS agent uses an adaptation of the basic form of MCTS (see Section 4). Because we do not know what moves the other players will do, and with the added difficulty of simultaneous play, we only save the first tuple of moves from a playout with its associated *won* and *played* counters. In essence, the tuple becomes the node, similar to Figure 6. We do playouts from the root, using random agents. This means that some tuples of moves may not be played at all when there are not enough playouts per turn. After each playout, the scores for the first tuple of moves done by the player representing the MCTS agent are updated. The MCTS algorithm does *plays* playouts, where *plays* can be chosen by the user. A playout is finished when a player has won, or the turn limit has been reached. When all playouts are finished, the tuple with the highest average win rate is delivered to the adjudicator. If multiple tuples have the same average win rate, one of these is chosen at random.

6.4 Minimax

We have two agents for minimax: one uses the minimax value, the other uses a maxiavg value (the maximum of the averages). For each phase of the next in-game year, these agents generate the Cartesian product of the valid orders for all units and fill a search tree with all the resulting tuples, where the root is the current phase, and each next phase comes on a new level. Then the agents calculate a fitness value for each leaf node and propagate it back to the root node. Finally, the tuple from the node on the first level with the highest fitness is delivered to the adjudicator. If multiple nodes have the same fitness one of the corresponding tuples is chosen at random.

The following is an example of how to calculate minimax for a three-player game with simultaneous moves using the maximin value. Let us consider the situation where each player

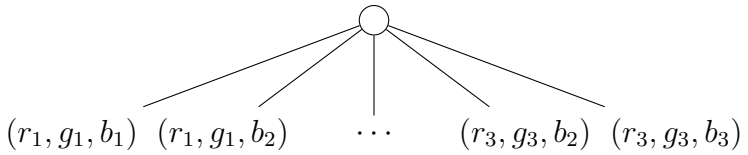

Figure 6: Example of a search tree.

(a)

b_1	g_1	g_2	g_3
r_1	$(6, -20, -4)$	$(-10, -11, -12)$	$(16, 16, -20)$
r_2	$(14, -16, -8)$	$(-6, 11, 8)$	$(10, -20, -20)$
r_3	$(-4, -18, -7)$	$(-11, 9, 10)$	$(-7, -13, -7)$

(b)

b_2	g_1	g_2	g_3
r_1	$(-17, -16, -6)$	$(-16, 0, -5)$	$(-7, 8, -20)$
r_2	$(-11, 9, -9)$	$(-7, -19, -14)$	$(-16, 9, -20)$
r_3	$(-15, 10, -6)$	$(-13, -18, -16)$	$(17, 13, 1)$

(c)

b_3	g_1	g_2	g_3
r_1	$(-17, -15, 10)$	$(-12, -1, -17)$	$(8, 14, -14)$
r_2	$(-4, 15, -10)$	$(-15, -14, 4)$	$(-14, -7, 14)$
r_3	$(15, -7, -13)$	$(4, 8, -9)$	$(7, 12, -3)$

Table 2: Payoff matrix. Player b chooses move: (a) b_1, (b) b_2, (c) b_3.

has three moves: $R = \{r_1, r_2, r_3\}, G = \{g_1, g_2, g_3\}$ and $B = \{b_1, b_2, b_3\}$. Figure 6 shows the tree representation for a one-level deep minimax algorithm. Assume Tables 2a, 2b and 2c form the payoff matrix (we use three tables for easier representation). For simplicity we consider a paranoid player: the player only looks at the best value he could get if the other players play against him and would be reading his mind. This is equivalent with turnbased maximin. For player r we then get the minimax tree in Figure 7. Note that it does not matter which of R's opponents goes first, as long as both of them play against R. As we can see, both G and B pick the move which, considering r's possible moves, leaves R with the lowest possible score.

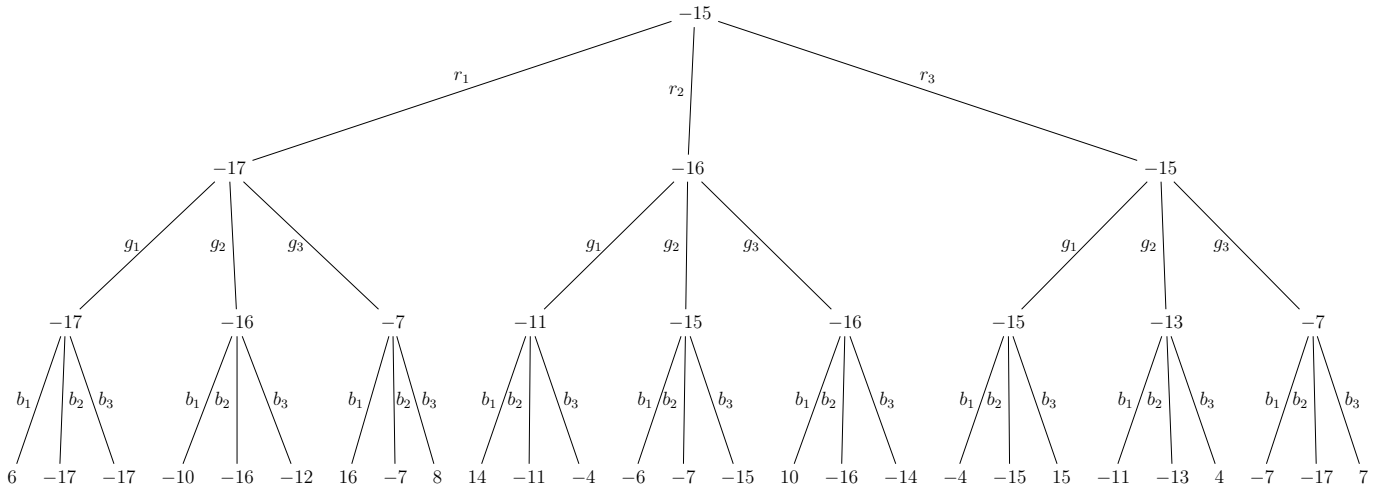

Figure 7: Example of a maximin search tree.

Table 3 shows the minimum score per move, that each player can be sure to receive. The maximum score that each player can be sure to receive is the maximum of their minimum scores, this is -15 for r (move r_3), -19 for g (move g_2) and -17 for b (move b_3). Now, if all players are paranoid, each player will play their move with the maximum score they can be sure to receive (r_3, g_2, b_3). Considering this set of moves r will score 4 points, g 8 points and b -9 points.

$\min(r_1) = -17$	$\min(g_1) = -20$	$\min(b_1) = -20$
$\min(r_2) = -16$	$\min(g_2) = -19$	$\min(b_2) = -20$
$\min(r_3) = -15$	$\min(g_3) = -20$	$\min(b_3) = -17$

Table 3: Minimum score for each move.

7 Experiments

We experimented with the boards in Figure 8, for players Red, Green and Blue, and the standard Diplomacy board (Figure 1). Each node in the figures represent a province, color-filled nodes indicate supply centers (where gray is neutral and the other colors are home supply centers for the players). To reduce complexity, all provinces are land provinces. The home supply centers contain an army for the corresponding player. The turn limits are as follows: *The Diamond* – 10 years, *The Triangle* – 12 years, *The Hexagon* – 18 years, *The Square* – 32 years.

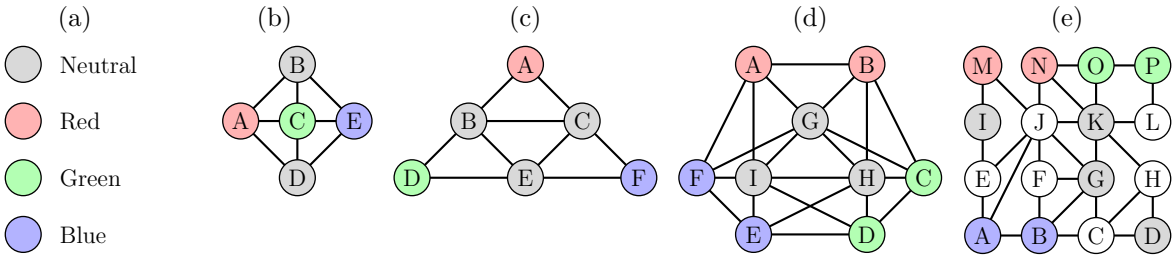

Figure 8: (a) Legend: colored nodes indicate a supply center and white nodes are provinces without supply center. Test maps: (b) *The Diamond*, (c) *The Triangle*, (d) *The Hexagon* and (e) *The Square*.

In our first attempts to experiment with minimax we determined that this method, without pruning, took far to long to be feasible for Diplomacy. This can be explained by taking a look at the branching factor: For now, lets ignore the retreat and adjustments phases. If we take a look at *The Diamond* we have 3 units, each having roughly 10 distinct orders (supports for other powers included). This means we have a branching factor 1000, which is 1000 after the first turn, one million after the second turn (one in-game year) and one billion after the third turn. This is already far more than we can process in reasonable

time. Even though pruning may drastically reduce the size of the tree, and possibly give a sensible move for this tiny map, for the standard board of Diplomacy this will still not be enough.

7.1 Analysing the maps

At first glance we can see that *The Triangle* and *The Hexagon* are fully symmetric, while *The Diamond* is symmetric for the red and blue players, and *The Square* does not contain any symmetry at all. We can also see that *The Hexagon* would not be possible on a two-dimensional board, because it is non-planar. *The Square* is the only map that can be considered similar to the map of standard Diplomacy, both in terms of supply centers and in terms of degree of the vertices.

We conducted an experiment to find the average winrates for each starting position, the results can be found in Tables 4, 5, 6 and 7. For the column *Random* we played 1000 games with three random agents. Note that some of the played games ended in a draw if the figures in this column do not add up to 1. For the *MCTS(plays)* columns used one MCTS(*plays*) agent and two random agents. Here *plays* is the number of playouts that the agent did for each turn. We played 1000 games for each starting position of the MCTS agents. The *Rel. increase* columns show the relative increase of winrates of the corresponding strategy compared to the random strategy.

From the tables we can see that for all of our maps the win rates for the MCTS strategies do not increase much after 500 playouts. This indicates that we are near the maximal potential of MCTS.

The Diamond

From Table 4 it follows that *The Diamond* is symmetric for red and blue. The win rates show that the starting position of green is worse than the starting positions of red and blue. However, the relative increases show that MCTS does not perform significantly better from any one position compared to the other positions. The win rates for the MCTS strategy do not increase much after 500 playouts, indicating that, at 500 playouts, our MCTS strategy is near its maximal potential on this map.

From the *Random Win rate* column we can see that roughly 3% of the played games ended in a draw. Interestingly, when the MCTS strategy is near its maximal potential it's not even close to winning all of the games. This is may be caused by the map having so few supply centers, resulting in a loss when the player fails to capture a supply center.

The Triangle

From Table 5 it follows that *The Triangle* is fully symmetric. For this map our MCTS strategy also appears to be near its maximal potential at 500 playouts, because the win rates for the MCTS strategy do not increase much after 500 playouts.

	Random	MCTS(100)		MCTS(500)		MCTS(1000)	
Power	Win rate	Win rate	Rel. increase	Win rate	Rel. increase	Win rate	Rel. increase
R	0.32	0.55	1.72	0.70	2.19	0.72	2.25
G	0.30	0.44	1.47	0.57	1.90	0.59	1.97
B	0.35	0.54	1.54	0.69	1.97	0.72	2.06

Table 4: Win rates for *The Diamond*.

From the *Random Win rate* column we can see that random play ended in a draw for around 47% of the played games. The MCTS strategy does come close to winning all of its games on this map, which is surprising, considering the results for MCTS on *The Diamond*.

	Random	MCTS(100)		MCTS(500)		MCTS(1000)	
Power	Win rate	Win rate	Rel. increase	Win rate	Rel. increase	Win rate	Rel. increase
R	0.18	0.74	4.11	0.89	4.94	0.92	5.11
G	0.18	0.74	4.11	0.89	4.94	0.92	5.11
B	0.17	0.69	4.06	0.87	5.12	0.90	5.29

Table 5: Win rates for *The Triangle*.

The Hexagon

From Table 6 it follows that *The Hexagon* is fully symmetric too. On this map the win rates for the MCTS strategy do not increase much after 500 playouts either, again indicating that, at 500 playouts, our MCTS strategy is near its maximal potential on this map.

From the *Random Win rate* column we can see that random play ended in a draw for around 3% of the played games. Again, the MCTS strategy near its maximal potential is not even close to winning all of the games. *The Diamond* and this map are similar in connectivity between provinces, in that from any one province, most other provinces can be directly reached, whereas *The Triangle* is much different in this regard, only being able to directly reach a few other provinces from any one province. From this and the results on *The Triangle* we conclude that maps with less connectivity require more tactical play to win.

	Random	MCTS(100)		MCTS(500)		MCTS(1000)	
Power	Win rate	Win rate	Rel. increase	Win rate	Rel. increase	Win rate	Rel. increase
R	0.31	0.65	2.10	0.77	2.48	0.78	2.52
G	0.33	0.61	1.85	0.76	2.30	0.74	2.24
B	0.33	0.60	1.82	0.69	2.09	0.75	2.27

Table 6: Win rates for *The Hexagon*.

The Square

From the relative increases in Table 7 it follows that *The Square* has no symmetry at all. The starting position of the red player appears to be far superior to the other starting positions. The likely cause for this is the close proximity of red to most of the other supply centers. The starting position of the green player is slightly worse than the blue starting position. We suspect this is caused by one of the red and green supply centers being neighbours, limiting the safe options for green during the first turns, while blue can move around freely without any danger. Our MCTS strategy appears to also be near its maximal potential around 500 playouts on this map. From the *Random Win rate* column we can see that random play ended in a draw for around 25% of the played games. For the red player, the MCTS strategy is close to winning all games with 500 playouts. Considering the win rates for random play, MCTS performs really well, even on the worst starting position.

	Random	MCTS(100)		MCTS(500)		MCTS(1000)	
Power	Win rate	Win rate	Rel. increase	Win rate	Rel. increase	Win rate	Rel. increase
R	0.41	0.86	2.10	0.92	2.24	0.94	2.29
G	0.13	0.54	4.15	0.73	5.62	0.77	5.92
B	0.20	0.75	3.75	0.85	4.25	0.90	4.50

Table 7: Win rates for *The Square*.

7.2 Analysing MCTS

For each of our own maps we conducted an experiment where we let the random, MCTS(100) and MCTS(*plays*) strategies play against each other, with $plays = 25, 50, \ldots, 1000$. We ran 167 games of all permutations of starting positions (summing up to a total of 1002 games). The results can be found in Figures 9a, 9b, 9c and 9d. From the graphs it follows that MCTS works as intended on all maps, we can see the expected form of a logarithmic function of the win rates for the MCTS strategies. We can also see that at 500 iterations, the MCTS agents still have not reached their full potential yet. As expected, the graphs for the MCTS agents cross at 100 iterations. Interestingly, after roughly 50 playouts, the random strategy only has a small decline in win rate on *The Diamond* and *The Hexagon* while the win rates are still relatively large. We assume that this is caused by how easy it is to win without tactical play on these maps. On *The Square* the win rate of MCTS looks like it is declining after 950 playouts. Altough we are not entirely sure why this is, we suspect that this is caused by the random agent having a string of luck, selecting good moves.

We also performed an experiment with the standard board of Diplomacy. Table 8 shows the average numbers of supply centers for this experiment. Here we played 1000 games with only random agents and 5 games with six random agents and one MCTS(*play*) agent for each starting position of the MCTS agent (a total of 35 games per MCTS strategy). Playing the 1000 games with only random agents took 3.5 to 4 minutes (using a desktop computer with an Intel® i7-4770K cpu), giving us a good indication of how many playouts we can

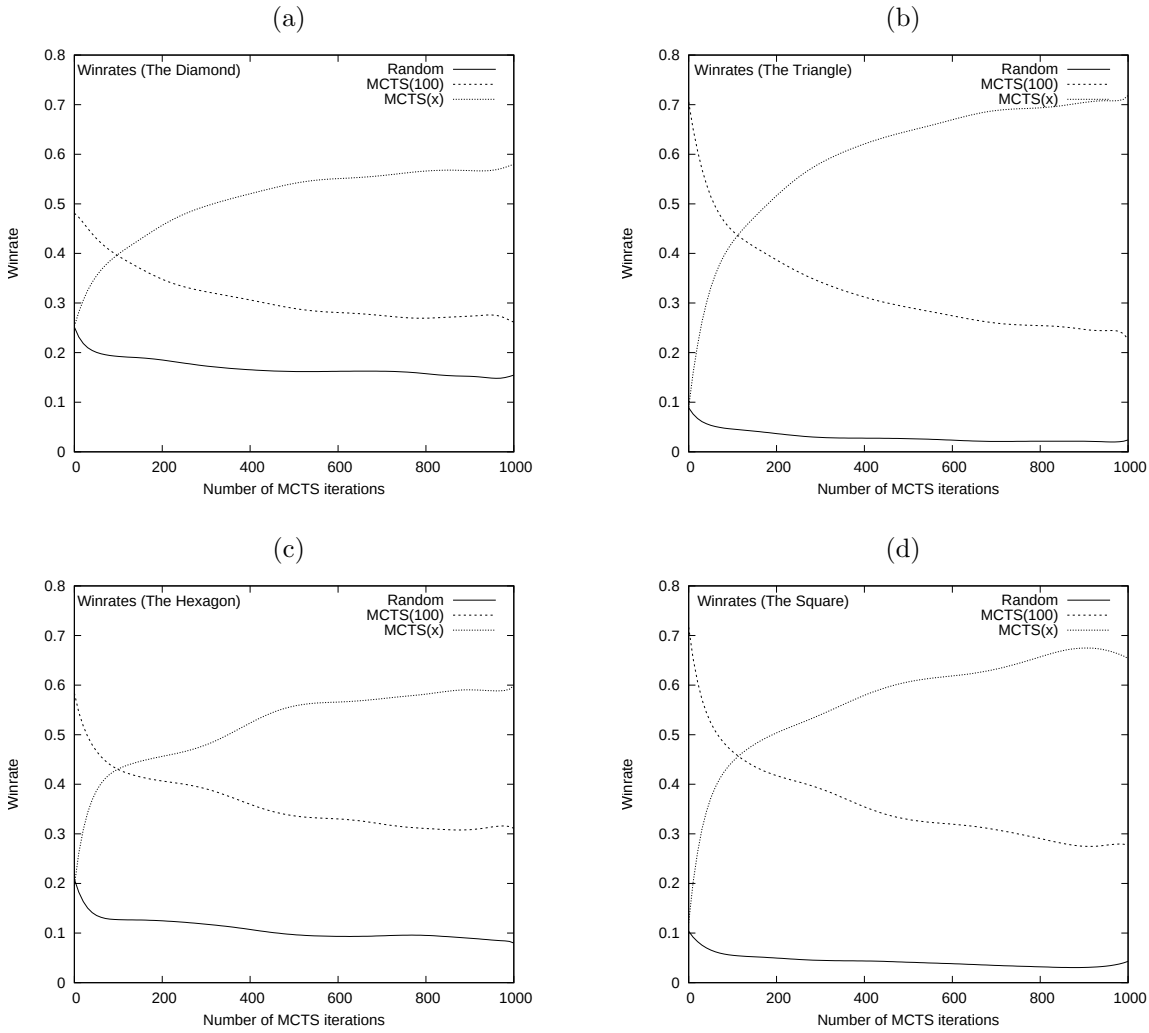

Figure 9: Winrates on (a) *The Diamond*, (b) *The Triangle*, (c) *The Hexagon* and (d) *The Square*.

perform to stay within the time limits specified in the game rules. From Table 8 it follows that, atleast for some powers, there is definately an increase in performance when using MCTS, however, for most powers MCTS needs many more playouts to win. This was not entirely unexpected, considering the size of the complete search sapce.

We were also interested to see whether or not the MCTS agents would be able to "learn" any strategic patterns. We did not find any proof that MCTS was able to recognize the openings adopted by the human players [20], but this was no suprise at all because many of the openings contain convoys, which we did not implement. However, we did find proof for recognizing simple patterns. At lower numbers of playouts we still find many silly orders such as trying to swap two of its own units or moving one of its units to another province where the agent has another unit which is holding. We gradually see those silly orders disappearing

Power	Random	MCTS(500)	MCTS(1000)
Austria	5.4	13.0	12.2
England	3.5	0.6	2.8
France	6.5	6.8	7.0
Germany	5.3	15.8	11.6
Italy	2.7	3.2	0.4
Russia	5.9	7.0	16.4
Turkey	4.7	1.4	3.4

Table 8: Average number of supply centers on the standard Diplomacy board.

when we increase the number of playouts and furthermore, see an increase in clever moves. The following example of a self-standoff occured on *The Square*:

- Army B, E – J
- Army B, G – J
- Army R, M – J

Here the blue player, a MCTS agent with only 100 playouts, uses two of its own units to defend a third province from another player. Furthermore, it appears as though the agent is aware of the strategic importance of a province, because we see self-standoffs mostly at provinces which we ourselfs consider important, for instance provinces with a lot of neighbours. We expected to see an increased amount of supports from MCTS agents compared to random agents, this was not the case though. In fact the MCTS agents did less supports, however, most of the supports that the random agents do are useless, whereas the MCTS agents make more usefull supports such as:

- Army R, J Supports Army R, N – K
- Army G, K Holds
- Army R, N – K

During the fall turn we can also see the MCTS agents holding units located at supply centers and trying to move to supply centers, if the agent does not control the supply center yet, taking the home supply centers of opponents more often than normal supply centers.

8 Conclusion and future work

We tested a basic variant of the MCTS algorithm against random agents for the game Diplomacy. The results have shown that MCTS performs well against random agents. The MCTS algorithm is decently fast. However, the quality of the moves is directly related to the amount of playouts, where more playouts increase the quality, but also increase the time it

takes to compute the agent its moves. Furthermore, a bigger or more complex map will also increase the runtime of the MCTS algorithm. Due to the size of the search spaces, even on small maps, minimax without pruning proved infeasible, with a single turn taking well over an hour.

8.1 Future work

MCTS should be compared with other agents, that are more clever than the random agent. If at all possible, a good idea would be to convert the MCTS agent to DipGame format. In addition to being able to compare the agent against other agents, this may also reduce the time it takes to come up with a solution (due to inefficiencies in our server).

There are several areas of improvement. First and foremost MCTS would probably greatly benefit from using abstract states of (part of) the maps. This could allow us to save a deeper search tree, which should greatly improve the performance in terms of win rate. UCT would then also make a great addition to the algorithm. It could also prove to be beneficial to, in some manner, remove obviously bad orders from the list of available orders, before the MCTS algorithm starts. Perhaps the succesful approach to Arimaa [21], where the enormous amount of possible moves could be ordered effectively, can be minimized.

Finally, the agent should be extended to make use of communications with other players. Negotiations are a vital part of Diplomacy and players will use it. An agent in a match with human players will be at a distinct disadvantage if negotiations are ignored.

References

[1] F. Hsu, "IBM's Deep Blue Chess Grandmaster Chips", IEEE Micro 19, pp. 70–81, 1999

[2] D. Silver, A. Huang, C. Maddison, D. Hassabis, "Mastering the Game of Go with Deep Neural Networks and Tree Search", Nature 529(7587), pp. 484–489, 2016

[3] The standard Diplomacy board:
`http://diplomacy.wikia.com/`,
accessed: June 2016

[4] The Diplomacy game rules:
`https://www.wizards.com/avalonhill/rules/diplomacy.pdf`,
accessed: June 2016

[5] L. Kruijswijk, "The Math of Adjudication":
`http://www.diplom.org/Zine/S2009M/Kruijswijk/DipMath_Chp1.htm`,
accessed: June 2016

[6] The DAIDE website:
`http://www.daide.org.uk/`,
accessed: June 2016

[7] A. Fabregues, S. Biec, C. Sierra, "Running Experiments on DipGame Testbed (Demonstration)", 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012), pp. 1481–1482, 2012

[8] S. Kraus, D. Lehmann, "Designing and Building a Negotiating Automated Agent", Computational Intelligence 11, pp. 132–171, 1995.

[9] D.E. Loeb, M.R. Hall, "Thoughts on Programming a Diplomat", Heuristic Programming in Artificial Intelligence, 3: The Third Computer Olympiad, pp. 123–145, 1992

[10] A. Shapiro, G. Fuchs, R. Levinson, "Learning a Game Strategy Using Pattern-Weights and Self-Play", Proceedings of The Third International Conference on Computers and Games (CG 2002), LNCS 2883, pp. 42–60, 2002

[11] The DumbBot source code:
`http://www.ellought.demon.co.uk/dipai/dumbbot_source.zip`,
accessed: June 2016

[12] F. Håård, "Multi-Agent Diplomacy: Tactical Planning using Cooperative Distributed Problem Solving", Master Thesis, Blekinge Institute of Technology, 2004

[13] J. Ribeiro, P. Mariano, L.S. Lopes, "DarkBlade: A Program That Plays Diplomacy", Progress in Artificial Intelligence: Proceedings of the 14th Portuguese Conference on Artificial Intelligence (EPIA 2009), LNCS 5816, pp. 485–496, 2009

[14] The Albert website:
`https://sites.google.com/site/diplomacyai/home`,
accessed: June 2016

[15] S.J. Russell, P. Norvig, "Artificial Intelligence: A Modern Approach", Prentice Hall, third edition, 2010

[16] C.B. Browne, E. Powley, D. Whitehouse, S. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton, "A Survey of Monte Carlo Tree Search Methods", IEEE Transactions on Computational Intelligence and AI in Games 4, pp. 1–43, 2012

[17] R. Coulom, "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search", Proceedings of the 5th International Conference on Computers and Games, (CG 2006), pp. 72–83, 2006

[18] B. Brügmann, "Monte-Carlo Go", Technical Report, Max Planck Institute of Physics, 1993

[19] S. Gelly, L. Kocsis, M. Schoenauer, M. Sebag, D. Silver, C. Szepesvári, O. Teytaud, "The Grand Challenge of Computer Go: Monte Carlo Tree Search and Extensions", Communications of the ACM, Volume 55, pp. 106–113, 2012

[20] The Diplomacy openings library,
`http://diplom.org/Online/Openings/`,
accessed: July 2016

[21] D.J. Wu, "Designing a Winning Arimaa Program", ICGA Journal 38, pp. 19–40, 2015