Automated Negotiations for General Game Playing

Dave de Jonge
Western Sydney University
Penrith, NSW 2751, Australia
d.dejonge@westernsydney.edu.au

Dongmo Zhang
Western Sydney University
Penrith, NSW 2751, Australia
d.zhang@westernsydney.edu.au

ABSTRACT

In this paper we present a new algorithm for negotiations in non-zero-sum games. Although games have been studied extensively, most game playing algorithms have been developed under the assumption that players do not communicate. Many real-world problems, however, can be modeled as non-zero-sum games in which players may mutually benefit if they coordinate their actions, which requires negotiation. The field of Automated Negotiations is another important topic in AI, but in this field one usually assumes that utility functions have explicit expressions and can therefore be calculated easily. Traditional approaches do not apply to domains in which the utility values are instead determined by the rules of a complex game. In this paper we aim to bridge the gap between General Game Playing and Automated Negotiations. Our algorithm is an adaptation of Monte Carlo Tree Search that allows players to negotiate. It is completely domain-independent in the sense that it is not tailored to any specific game. It can be applied to any non-zero-sum game, provided that its rules are described in Game Description Language.

Keywords

Automated Negotiations, General Game Playing, Monte Carlo Tree Search, Non-zero-sum Games

1. INTRODUCTION

Games are important in Artificial Intelligence because they provide controlled environments with clear rules. They can be seen as simplified metaphors for real-world problems. In a sense, any multiagent system in which each agent has its own private goals that may be conflicting with the other agents' goals can be seen as a game. Most research on algorithms for games however, has focused on zero-sum games, such as Chess, Checkers and Go. This is striking, because many real-world problems are better modeled as non-zero-sum games. One can think for example of a system of self-driving cars that negotiate with each other which car will take which route, in order to prevent traffic jams. Also, the whole idea of a market economy is essentially a non-zero-sum game; each participant in the economy has its own personal goals and by exchanging goods and services with one another each individual participant may benefit.

Although traditional game-playing algorithms such as Minimax and Monte Carlo Tree Search can be easily adapted for non-zero-sum games, they do not allow the agent to negotiate its actions with its opponent, and may therefore yield inefficient outcomes. Typical examples of games where individual play is inefficient are the (Iterated) Prisoner's Dilemma [1], the Centipede Game [25], and the Dollar Auction [29]. If players were allowed to negotiate and make binding agreements in such games the outcomes would improve for each of them.

The field of Automated Negotiations is another important field of research within Artificial Intelligence. The domains investigated in Automated Negotiations, however, are often of a much simpler nature than traditional games. One usually assumes that the negotiators make proposals for which the exact utility values can be determined quickly [2]. Little attention has been given to negotiation settings in which determining the utility value of a deal is itself a hard problem. The preferences of the agent's opponents on the other hand, are often assumed to be completely unknown. This is in sharp contrast to Game Theory, in which reasoning about one's own utility and the opponent's utility is paramount. Furthermore, one usually assumes the negotiation algorithms do not require any domain knowledge or reasoning at all, or that all such knowledge is hardcoded in the algorithm.

Arguably the best-known example of a complex game that does involve negotiations is the game of Diplomacy [4]. A number of negotiating agents have been developed for this game, but they highly depend on details specific for Diplomacy and are therefore hard to generalize to other settings.

The aim of this paper is to bridge the gap between Automated Negotiations and Games, in a completely generic way. We present a new algorithm for non-zero-sum games that applies negotiations and that is re-usable because it is domain independent. In order to achieve this, we base our algorithm on Monte Carlo Tree Search, which is one of the most commonly used algorithms in the field of General Game Playing (GGP). The field of GGP studies algorithms for game playing agents, under the restriction that the rules of those games are only known at run-time. Therefore, when developing a GGP agent, one cannot use any game-specific heuristics. Although this makes it much harder to write strong players, the advantage is that the same algorithm can be re-used for any game.

Appears in: *Proc. of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017), S. Das, E. Durfee, K. Larson, M. Winikoff (eds.), May 8–12, 2017, São Paulo, Brazil.*

The rest of this paper is organized as follows. In Section 2 we give an overview of existing work on Automated Negotiations and GGP. In Section 3 we formally describe the problem we aim to solve and introduce notation. In Sections 4 and 5 we give brief introductions to Monte Carlo Tree Search and Automated Negotiations respectively. In Section 6 we then present our new algorithm. In Section 7 we present the results of our experiments, and finally, in Section 8 we discuss future work.

2. RELATED WORK

The earliest work on Automated Negotiations has mainly focused on proving formal properties of idealized scenarios. A seminal paper of this type is by Nash [21] in which it was shown that under certain axioms the rational outcome of a bilateral negotiation is the solution that maximizes the product of the players' utilities. Many papers have been published afterwards that generalize or adapt some of these axioms. A non-linear generalization has been made for example in [7]. A general overview of such theoretical studies is made in [28].

In later work focus has shifted more towards heuristic approaches for domains where one cannot expect to find any formal equilibrium results, or where such equilibria cannot be determined in a reasonable amount of time. However, such works often still make many simplifying assumptions. Important examples are [5] and [6]. They propose a strategy that amounts to determining for each time t which utility value should be demanded from the opponent (the *Aspiration Level*). However, they do not take into account that it may be hard to find a contract that indeed yields that aspired utility value. They assume that such a contract always exists, and that the negotiator can find it without effort. Another commonly made assumption is that the utility functions are linear and that each negotiator knows the explicit expression of its own utility function. This was the case, for example, in the first four editions of the Automated Negotiating Agent Competition (ANAC 2010-2013) [2].

Recently, more attention has been given to more realistic settings in which the number of possible deals is very large so that one needs to apply search algorithms to find good deals to propose, and where utility functions are non-linear [19, 11, 20]. Although their utility functions are indeed non-linear over the vector space that represents the set of possible contracts, the value of any given contract can still be calculated quickly by solving a linear equation, which is not always possible in a real-world setting. The idea of complex utility functions was taken a step further in [13], in which determining the value of any contract is NP-hard.

Search algorithms such as Genetic Algorithms (GA)[14, 23] and Simulated Annealing[11, 19] have been used to search for good proposals in complex domains. Unfortunately, GAs cannot be applied straightforwardly to games, because the search space is not closed under the two major operators of GA ('mutation' and 'crossover'). When applying these operators to a legal sequence of joint actions (see Section 3), the result will generally be an illegal sequence. A similar problem occurs with Simulated Annealing.

An important example of negotiations where determining the utility value of a deal is a hard combinatorial problem is the game of Diplomacy. In this domain the players' utility functions are not directly defined in terms of the agreements they make, but more indirectly through the moves they make in the game. The players negotiate with one another about which moves each will make, which in turn influences the outcome of the game. Determining the effect of an agreement on the player's final utility is a hard problem that involves Game Theory and Constraint Satisfaction. Pioneering work on negotiations in Diplomacy was presented in [26] and [17]. New interest in Diplomacy as a test-bed for negotiations has sparked with the introduction of the DipGame framework [4] for the development of Diplomacy agents for scientific research. Several negotiating Diplomacy players have been developed on this platform [8, 3, 12].

General Game Playing is a relatively new topic. Although earlier work has been done, it only started to draw widespread attention in the AI community after the introduction of Game Description Language (GDL) [18] and the introduction of the annual AAAI GGP competition in 2005 [10]. GDL allows one to write down the rules of a game as a machine-readable logic program. Common techniques applied by GGP players are Minimax [30], Alpha-Beta Pruning [15] and Monte Carlo Tree Search (MCTS) [16]. FluxPlayer [27]—the winner of the 2006 AAAI GGP competition—applies an iterated deepening depth-first search with alpha-beta pruning, and uses Fuzzy logic to determine how close a given state is to the goal state. Cadia Player [9]—the winner in 2007, 2008, and 2012—is based on MCTS, extended with several heuristics to guide the rollouts so that they are better informed and hence give more realistic results, and also the winner of the 2014 competition, Sancho,[1] and the winner of 2015, Galvanise,[2] apply variants of MCTS.

3. PROBLEM DESCRIPTION

In this section we introduce the problem we are tackling. We consider a setting in which two agents play a game that takes place over multiple rounds. In each round each agent chooses an action from a finite set of legal actions. However, before selecting their actions, the players have the opportunity to negotiate which strategies to follow. The agents only receive the rules of the game at run-time, so their implementations cannot use any game-specific heuristics.

3.1 Games

DEFINITION 1. *A **game** G (for 2 players) is a tuple $\langle Ag, \mathcal{A}, W, w_0, T, L, u, U \rangle$, where:*

- $Ag = (\alpha_0, \alpha_1)$ *is a pair of **agents** (or **players**).*
- $\mathcal{A}$ *is a pair $(\mathcal{A}_0, \mathcal{A}_1)$ where each $\mathcal{A}_i$ is the finite set of **actions** (or **moves**) of agent α_i.*
- W *is a finite, non-empty set of **states**.*
- $w_0 \in W$ *is the **initial state**.*
- $T \subset W$ *is the set of **terminal states**.*
- $L = (L_0, L_1)$*, where each L_i is the **legality function** of α_i, which assigns to each non-terminal state a nonempty set of legal actions $L_i : (W \setminus T) \to 2^{\mathcal{A}_i}$.*
- $u : (W \setminus T) \times \mathcal{A}_0 \times \mathcal{A}_1 \to W$ *is the **update function** that maps each non-terminal state and action-pair to a new state.*
- $U = (U_0, U_1)$ *where each U_i is the **utility function** of player α_i, which assigns a utility value to each terminal state: $U_i : T \to \mathbb{R}^+$.*

[1] http://sanchoggp.blogspot.co.uk/2014/05/what-is-sancho.html

[2] https://bitbucket.org/rxe/galvanise_v2

We say an action $a \in \mathcal{A}_i$ is **legal** for α_i in w iff $a \in L_i(w)$.

Informally, a *turn-taking* game is a game in which in every turn only one player (the *active player* of that turn) makes a move. Here however, we follow the standard convention in GGP that players always make moves simultaneously. This is not a restriction, because any turn-taking game can be modeled equivalently as a game with simultaneous moves by adding a dummy move to the game and assuming that the non-active player makes that dummy move instead of making no move at all. This dummy move is usually referred to as 'noop'. Therefore, we formally define turn-taking games as follows.

DEFINITION 2. *A game for 2 players is called a* ***turn-taking game*** *if we can partition the set of non-terminal world states into two subsets:* $(W \backslash T) = W_0 \cup W_1$, $W_0 \cap W_1 = \emptyset$, *such that for all* $i \in \{0, 1\}$ *we have:*

$$\text{If } w \notin W_i \text{ then } L_i(w) = \{noop\}$$

If $w \in W_i$ *we say that* α_i *is the* ***active player*** *of* w, *while the other player is called the* ***non-active player*** *of* w.

This definition says that in every state either α_0 or α_1 is the active player, and that the non-active player always has exactly one legal move, called *noop*. To keep the discussion simple we only explain our algorithm for turn-taking games. However, the algorithm we have implemented works for non-turn-taking games as well. In the rest of this paper we will always be talking about turn-taking games, unless specified otherwise. We will use the notation $u(w, a)$ as a shorthand for $u(w, a, noop)$ or $u(w, noop, a)$ where a is an action of the active player of w.

3.2 Client-Server Model

In this paper we assume the agents that play the game are implemented as clients in a client-server architecture. In order to play a game, the two agents need to connect to the server. The server then sends a message to each player containing the description of some game, written in GDL, which encodes the components of Def. 1. The agents are given some initial time to parse the game description and initialize their algorithms. Next, the game starts when the server sends a message to each player indicating the initial world state w_0. In general, whenever the server sends a message with some non-terminal state w_r each player α_i must reply with a message containing its next move $a_{r,i} \in L_i(w_r)$, after which the server sends a new message with the new state w_{r+1}, which is determined by the game's update function and the actions chosen by the players: $w_{r+1} = u(w_r, a_{r,0}, a_{r,1})$. This process repeats until the state sent by the server is a terminal state $w_T \in T$. Each player α_i then obtains the utility value $U_i(w_T)$ corresponding to that terminal state. If a player does not respond to the server within a specified deadline, or responds with an illegal action, the server will instead pick a random legal action for that player. We sometimes say that the game is *in round* r or that the game is *in state* w_r, meaning that the last message sent by the server contained the state w_r.

3.3 Sequences

A pair of actions $p = (a_0, a_1) \in \mathcal{A}_0 \times \mathcal{A}_1$, one for each player, is called a **joint move**. A joint move (a_0, a_1) is **legal** in w, if a_0 is legal for α_0 in w and a_1 is legal for α_1 in w.

DEFINITION 3. *Let* w_r *be a state and let* $s = (p_r, p_{r+1}, \ldots p_{r+n-1})$ *be a sequence of joint moves. We say that* s *is a* ***legal sequence starting at*** w_r, *iff there exists a sequence of states* $(w_r, w_{r+1} \ldots w_{r+n})$ *such that for each* $k \in [r \ldots r+n-1]$ *the joint move* p_k *is legal in* w_k *and* $w_{k+1} = u(w_k, p_k)$.

We use the notation $s_{k,i}$ to denote the action of player α_i in the joint move p_k of the sequence s. That is: $p_k = (s_{k,0}, s_{k,1})$. The state w_{r+n} in this definition is called the **resulting state** of the sequence s.

3.4 Negotiations

So far, our description has been no different from any other set-up for (general) game playing. What we want, however, is to allow the players to negotiate. Therefore, after the server has sent the current world state w_r to the players and before the players reply with their next actions, we allow the two players to exchange messages with each other according to some negotiation protocol. These negotiation messages are of the type $propose(s)$ or $accept(s)$ in which s can be any legal sequence starting at w_r, of any length n. If one player proposes a sequence s and the other accepts it, then both players must obey it. This means that for each w_k with $r \leq k < r+n$ each player α_i must play the move $s_{k,i}$. However, even if the players have already agreed to play some sequence s, the players may continue to negotiate and agree on a new sequence s'. In that case the first agreement is discarded, and instead the players are only required to obey the newly agreed sequence s'.

The details of the negotiation protocol are irrelevant for this paper. It could be the Alternating Offers protocol [24], but it may just as well be any other bilateral protocol.

We regard every round of the game as a separate negotiation session. After all, in each round the state w_r of the game is different, and therefore the sequences the agents may propose to each other are different.

Although we only allow players to propose linear sequences of moves, everything explained works just as well for more complex types of proposals such as tree-shaped structures. However, it would take up too much space to give a rigorous definition of such deals, and to explain how to assign utility values to such deals. Furthermore, the current implementation of our algorithm only allows linear sequences because otherwise we would first need to establish a communication language that allows the agent to express more complex proposals. There is no fundamental problem in doing so, but we leave this as future work.

To conclude this section, we would like to stress two important points. Firstly, we emphasize that the agreements the players make are considered binding. Once an agreement is made, the players cannot deviate from it unless they make a new agreement. We do not discuss how to enforce such agreements in a real-world setting. We simply assume the game server will enforce the agreements in the same way that it enforces the rules of the game; if a player makes a move that is inconsistent with the last agreement made, then the server will ignore it and instead select a move that does obey the agreement. Secondly, we should stress that we are assuming the players to be selfish. Each player is only interested in maximizing its own utility, and is not interested in maximizing any form of 'social utility'. Therefore, a player

is only willing to accept a proposal if it estimates that this will not decrease its utility.[3]

4. MONTE CARLO TREE SEARCH

In this section we give a quick introduction to Monte Carlo Tree Search (MCTS). For more information we refer to [16]. We should note that MCTS is an algorithm for games *without* negotiations. In Section 6 we describe how we have adapted it for games with negotiations.

MCTS is an anytime algorithm that, given the current state of a game, returns an estimation of the best move to make. This is especially useful for games in which the state space is too large to be explored exhaustively. Given a game G, MCTS iteratively generates a tree in which every node ν is labeled with a state w_ν. If ν' is a child of ν it means that the active player of w_ν has a legal action a such that $w_{\nu'} = u(w_\nu, a)$. The tree is initialized with a single root node that represents the initial state w_0. The algorithm cycles through the following steps, which are further explained in the next subsections.

1. Selection: select a branch of the tree that starts at the root and ends at some leaf node ν_l.
2. Rollout: randomly pick a legal sequence of joint moves that starts at the state w_l corresponding to the node ν_l and that ends at some terminal state w_T.
3. Update: for each player α_i and each node ν in the branch selected in step 1 update the Average Rollout Scores $\widetilde{U}_{i,\nu}$.
4. Expansion: generate children for ν_l.

Whenever we refer to a *leaf node*, we mean a node for which no children have *yet* been generated. This should not be confused with a *terminal node*, which is a node that represents a terminal state and therefore cannot have any children.

4.1 Selection

Each cycle starts with the selection of a branch of the tree from the root to some leaf node ν_l which is the 'best' according to some heuristic. This is done by starting at the root, then selecting the child ν of the root with the highest heuristic value, then selecting the child ν' of ν with the highest heuristic, etcetera, until a leaf node is reached.

A purely greedy algorithm would simply use the Average Rollout Score $\widetilde{U}_{i,\nu}$ (explained below) for the active player of w_ν as the heuristic. However, this usually does not work well. Therefore, a commonly used heuristic is the UCT heuristic [16], which adds an 'exploration term' uct_ν to the Average Rollout Score. A node has a high exploration term if it is has been selected relatively few times in comparison to its siblings.

4.2 Rollout

Having selected a branch that ends at some leaf node ν_l the algorithm randomly generates a legal sequence of joint moves that starts at the state w_l corresponding to ν_l and ends at some terminal state[4] w_T. It then determines for each player α_i its utility $U_i(w_T)$ for this terminal state.

[3]We say 'estimates' here, because the players are only assumed to be bounded rational.

[4]This of course only works if we are guaranteed that no infinite sequences of legal joint moves can exist.

4.3 Update

After the rollout, for every node ν in the selected branch and every player α_i the *Average Rollout Score* $\widetilde{U}_{i,\nu}$ is updated, to ensure it satisfies the following formula:

$$\widetilde{U}_{i,\nu} = \frac{1}{N_\nu} \sum_{j=1}^{N_\nu} U_i(w_j)$$

where N_ν is the total number of times that ν was in the selected branch among all previous iterations, and w_j is the terminal state that resulted from the rollout in the j-th iteration in which ν was in the selected branch.

A key feature of MCTS is the fact that if the algorithm is run long enough, the Average Rollout Scores of the nodes will converge to their theoretical Minimax values.

4.4 Expansion

In the last step of the cycle children are generated for the leaf node ν_l. If α_j is the active player of w_l, then the algorithm will generate one child ν_a for each action $a \in L_j(w_l)$. Each child ν_a will be labeled with the state w_a that satisfies: $w_a = u(w_l, a)$.

5. TRADITIONAL NEGOTIATIONS

In this section we give a short introduction to traditional Automated Negotiations. In a typical, classical negotiation domain two agents α_0 and α_1 are bargaining to agree on some contract. The space of all possible contracts is called the *contract space*. Each agent α_i has its own utility function U_i that maps each possible contract to a utility value, which is usually a positive real number. The utility functions are private in the sense that α_0 does not know the utility function U_1 of its opponent, and vice versa. For a contract x we refer to the vector $\vec{U}(x) = (U_0(x), U_1(x))$ as the *utility vector* of x and we call the set of all utility vectors the *utility space*.

The agents have a fixed amount of time to exchange proposals according to some protocol. That is: an agent may propose a contract x, and then the other agent may either accept the proposal, or make a counter proposal. If a proposal x is accepted by the other agent, then both agents receive their corresponding utility values ($U_0(x)$ and $U_1(x)$ respectively). However, if no proposal is accepted before the deadline t_{dead}, each agent α_i will receive a pre-defined amount of utility which is known as its *Reservation Value* rv_i. Clearly, a rational negotiator would never accept any proposal that yields an amount of utility less then its Reservation Value. Just like the utility functions, the Reservation Values are usually assumed to be private.

A typical strategy for a negotiating agent α_i is to apply a time based concession strategy. This means it has a time-dependent function, called its *Aspiration Level*, $asp_i(t)$. This function decreases over time from some initial value to some final value which is greater than or equal to its Reservation Value. Then, if at some time t it is α_i's is turn to make a proposal it picks a contract x from the contract space for which $U_i(x) = asp_i(t)$ holds, and proposes that contract to its opponent. When α_i receives a proposal x at time t, it will accept that proposal if $U_i(x) \geq asp_i(t)$ holds.

Note that in this paper we are assuming players have full information about each other's utility functions, as these are given in the GDL description. However, in our case the util-

ity functions are not defined directly over the contract space, but instead over the set of terminal states of some game G, while the contracts are legal sequences of joint moves of G. In the case that a sequence s results in a *terminal* state w_s we can assign utility values $U_i(s)$ to the sequence, which equal $U_i(w_s)$. In case the resulting state is not terminal we can, in theory, assign a value $U_i(s)$ to s that equals the equilibrium value for player α_i in the subgame that starts at the state w_s. In practice however, this may be hard to calculate.

Furthermore, we should note that even though for some contracts we can determine the utility values exactly, we generally cannot determine a player's Reservation Values exactly (see Section 6.5), because that would require the agent to explore the search tree exhaustively. This is important, because for a negotiator the value of a deal is determined by the difference between its utility and the negotiator's Reservation Value, rather than just the pure utility value.

6. MCTS FOR NEGOTIATIONS

We are now ready to present our new algorithm for bilateral negotiations over 2-player games. Just like regular MCTS it generates a tree in which every node represents a possible future state. However, in our case in every round of the game the players negotiate. This means that each tree node also represents a possible future negotiation session, and therefore, for every node ν, we do not only need to calculate the Average Rollout Scores, but also a pair of Reservation Values, $rv_{\nu,i}$, and a pair of *Negotiation Values*, $nv_{\nu,i}$ (see Section 6.4).

The Negotiation Values of a node ν_w represent the utility values the players may reasonably expect to achieve when negotiating while the game is in state w, while the Average Rollout Scores represent the utility values the players may expect if they do not negotiate.

Again like regular MCTS, our algorithm applies a rollout-procedure to randomly sample the search space. In our case, every legal sequence generated by this procedure is stored as a potential proposal. For such proposals we can exactly calculate both players' utility values, because the resulting state of a rollout sequence is terminal.

Similarly, every branch from the current root to any other tree node represents a legal sequence of joint moves that may be proposed. However, such a legal sequence does not necessarily result in a terminal state. Instead, we can use the Average Rollout Scores of the last node of that branch as the proposal's utility vector, although that would only be an approximation of the true utility the respective players would receive from such a deal.

In the rest of this section we will describe the algorithm as if it is running on agent α_0. The opponent α_1 may be running the same algorithm, but may just as well be running any other algorithm, or may even be human.

6.1 Main Algorithm

At the beginning of each new round, right after our agent has received the new state w_r from the server, the algorithm starts by finding the node ν_r in the tree that represents the state w_r, and pruning all nodes in the tree that are not in the subtree under ν_r. The node ν_r is now the new root of the tree (assuming ν_r was already generated in the tree during earlier rounds).

Next, it executes Algorithm 1. This algorithm contains two nested loops. Inside the outer loop, it first calls the function selectBranchNego() to select a branch ending at some leaf node ν_l. Unlike regular MCTS, this selection is not made based on the Average Rollout Scores, but instead is based on the Negotiation Values (see Section 6.2).

The inner loop runs for half a second. Inside the inner loop a subtree is generated under ν_l, using regular MCTS. Note that this loop starts by selecting a branch ending at a leaf node ν'_l underneath ν_l. Of course, in the first iteration of the inner loop ν_l itself is already a leaf node, so ν'_l will be equal to ν_l. In the following iterations however, the node ν_l has been expanded, so ν_l is no longer a leaf.

Algorithm 1 MCTS-NEGO

Require: w_r, t_{dead}
1: $t \leftarrow$ getCurrentTime()
2: **while** $t < t_{dead}$ **do**
3: $\quad\nu_l \leftarrow$ selectBranchNego()
4: $\quad t'_{dead} \leftarrow t + 500$ //loop for 500 ms.
5: $\quad$**while** $t < t'_{dead}$ **do**
6: $\quad\quad\nu'_l \leftarrow$ selectBranchMCTS(ν_l)
7: $\quad\quad s \leftarrow$ rollout(ν'_l)
8: $\quad\quad$update(ν'_l, s)
9: $\quad\quad$expand(ν'_l)
10: $\quad\quad t \leftarrow$ getCurrentTime()
11: $\quad$**end while**
12: $\quad$receiveIncomingMessages()
13: $\quad$updateNegoValuesAndResValues(ν_l, s)
14: $\quad asp_0^0 \leftarrow$ calculateMyAspLevel(t)
15: $\quad asp_0^1 \leftarrow$ calculateOpponentAspLevel(t)
16: $\quad$acceptOrPropose(asp_0^0, asp_0^1)
17: **end while**
18: $a \leftarrow$ selectMoveToMake()
19: **return** a

The functions that are called inside the inner loop work exactly as in normal MCTS. When the inner loop has finished, the Average Rollout Scores of ν_l should have been determined accurately enough so that we can use them to update the Reservation Values (as explained in Sec. 6.5) and Negotiation Values (Sec. 6.4) of all its ancestors. The algorithm next calculates the player's aspiration levels, and decides which deal to propose to the opponent, or which received proposal to accept.

6.2 Selecting Branches

Algorithm 1 applies two functions to select the next branch to explore. Given some node ν the function selectBranchMCTS() selects a branch from ν to some leaf node ν'_l in the subtree under ν, in the same way as regular MCTS. The function selectBranchNego(), however, selects a branch based on the Negotiation Values, rather than on the Average Rollout Scores. That is: it starts with the root node and then, if the active player is α_i, selects the child ν of the root for which the sum $nv_{\nu,i} + uct_\nu$ is maximal. This is repeated, until a leaf node is reached.

Using the Negotiation Values to select a branch rather than the Average Rollout Scores allows us to focus the search on that part of the tree where we expect to find more contracts that are beneficial to both players. However, in order to correctly calculate the Negotiation Values of ν_l, we need to determine its Average Rollout Scores, and therefore we need to generate a subtree under ν_l using regular MCTS.

6.3 Aspiration Levels

At regular intervals our agent needs to decide which proposals to propose or accept. Traditionally, such decisions are made based on a time-dependent Aspiration function asp_i [5]. The classical notion of an Aspiration Level however, was invented for domains in which all possible contracts yield Pareto-optimal utility vectors. In our case, contracts may not always be Pareto-optimal, and the negotiators may not know whether they are Pareto-optimal or not, because the game is complex and agents have bounded rationality.

Therefore, we use an alternative that was introduced in [13], in which α_0 uses two time-dependent Aspiration functions asp_0^0 and asp_0^1. Here, asp_0^0 is a decreasing function that represents the amount of utility α_0 demands from α_1, while asp_0^1 is an increasing function that represents the amount of utility α_0 aims to offer to α_1.

For a given time t a contract x is *selfish enough* if $U_0(x) \geq asp_0^0(t)$ and it is *altruistic enough* if $U_1(x) \geq asp_0^1(t)$. Agent α_0 only proposes contracts that are selfish enough but, if possible, proposes contracts that are also altruistic enough. If it has discovered more than one such contract, it will propose the one for which $U_0(x)$ is maximal. If, on the other hand, it has not discovered any such contract, it will propose the contract for which $U_1(x)$ is maximal among those contracts that are selfish enough. If no contract is known to the agent that is selfish enough, it will not propose anything.

Whenever α_0 receives a proposal from α_1 that is selfish enough for α_0, then α_0 will accept it.

6.4 Negotiation Values

When implementing a negotiating agent, an important strategic decision one needs to make is the question how far to concede. That is: what should be the value of $asp_0^0(t_{dead})$, where t_{dead} is the deadline of the negotiation. One could simply choose this value to be equal to the Reservation Value rv_0, but this is a weak strategy, because any opponent could easily exploit it by conceding as little as possible. On the other hand, choosing this value to be very high is a risky strategy, because if the opponent does the same, it is likely that no deal will be made because the contract space does not contain any contract that yields enough utility to both agents. Therefore, we need to determine which utility value we can realistically expect α_0 to obtain and concede no further than that.

Note that if the utility space is convex, then the solution to this problem is already given by Nash [21]. According to the Nash Bargaining Solution both negotiators should concede towards the contract x that maximizes the product $U_0(x) \cdot U_1(x)$. However, in our case the contracts are sequences of joint moves, which form a discrete, and therefore non-convex, space. This could be solved by allowing the agents to also negotiate 'lotteries' of sequences. That is: the agents agree to flip a (biased) coin, and if the outcome is 'heads' they will play the sequence s, while if the outcome is 'tails' they will play sequence s'. If we allow the coin to be biased with any probability P, then the utility space becomes convex. However, the problem with this solution is that one needs to trust that the coin is indeed exactly biased with the agreed probability P. One would need some external source of randomness that is trusted by both agents. Such a source is not always available.

In this paper we propose another solution. In our solution, the players only make deterministic agreements. However, player α_0 itself will flip a coin in order to choose whether it will insist on some agreement s with high utility, or concede to an alternative agreement s' with lower utility. The important difference here is that the coin is only used internally by α_0, so there is no problem with trust.

	$Persist$	$Yield$
$Persist$	rv_0, rv_1	A, B
$Yield$	C, D	$\frac{1}{2}(A+B), \frac{1}{2}(C+D)$

Table 1: Payoff matrix of G_w

Let us explain this with an example. Suppose the agents are playing some game G, which is in some state w. Let us assume that for both players the Reservation Value in this state is 0, and that they have to choose between two Pareto-optimal contracts x and y with corresponding utility vectors $\vec{U}(x) = (60, 40)$ and $\vec{U}(y) = (40, 60)$ respectively. We see that player α_0 has the choice to either *persist* and demand at least 60 utility points (meaning that he is only willing to accept contract x), or to *yield* and also accept to receive only 40 utility points (meaning he is also willing to accept contract y). The opponent α_1 has exactly the same two options, but with the roles of x and y reversed.

The choice which strategy to follow, to persist or to yield, may itself also be seen as a game, which we refer to as G_w. If both players persist, the negotiations fail, and both players will receive 0 utility points. If one player persists and the other yields, then the persisting player receives 60 points, while the other receives 40 points. If both players yield, the outcome depends on how fast the players' Aspiration Levels drop. The player who concedes fastest will receive 40 points, while the other will receive 60 points. Here we will simply assume that in that case there is a 50% chance that they will agree on contract x, and 50% chance it will be y, so the expected utility for both players is 50.

In general, if we have contracts x and y with utility vectors $\vec{U}(x) = (A, B)$ and $\vec{U}(y) = (C, D)$ and the Reservation Values for the players are rv_0 and rv_1, then choosing the concession strategy can be modeled as a game G_w with a payoff matrix as displayed in Table 1. One can easily calculate (see e.g. [22]) that the Mixed Strategy Nash Equilibrium of G_w is given by:

$$P_0 = \frac{D - B}{B + D - 2 \cdot rv_1} \qquad P_1 = \frac{A - C}{A + C - 2 \cdot rv_0} \quad (1)$$

where P_i represents the probability that player α_i will play *persist*. Agent α_0 can now determine its final Aspiration Level $asp_0^0(t_{dead})$ by flipping a coin and setting it equal to A with probability P_0 and to C with probability $1 - P_0$.

Furthermore, α_0 can now also calculate the expectation values of the utility both agents will receive from negotiating in state w.

Definition 4. *Given a game G and a non-terminal state w of G, we define the* ***Negotiation Value*** *$nv_{w,i}$ for player α_i as the expected utility of α_i in the game G_w. For terminal states it is defined as the exact utility value of that state: $nv_{w_T,i} = U_i(w_T)$.*

If we apply this to our example, then we find that $P_0 = P_1 = \frac{1}{5}$, and for the Negotiation Values we find:

$$nv_{w,0} = \frac{1}{5} \cdot \frac{1}{5} \cdot 0 + \frac{1}{5} \cdot \frac{4}{5} \cdot 60 + \frac{4}{5} \cdot \frac{1}{5} \cdot 40 + \frac{4}{5} \cdot \frac{4}{5} \cdot 50 = 48$$

$$nv_{w,1} = \frac{1}{5}\cdot\frac{1}{5}\cdot 0 + \frac{1}{5}\cdot\frac{4}{5}\cdot 40 + \frac{4}{5}\cdot\frac{1}{5}\cdot 60 + \frac{4}{5}\cdot\frac{4}{5}\cdot 50 = 48$$

We see that both players can expect to receive 48 utility points. Interestingly, this is lower than the expected outcome in the traditional approach where the players negotiate lotteries. In that case, according to Nash, both players would expect to receive 50 utility points. The reason for this difference is that the Nash solution calculates the expected outcome under the assumption that the negotiations will succeed. However, in our case, there is always the possibility that negotiations fail, because both players may choose to persist. Of course, this means that if the players do have access to a trusted coin, then it is preferable for them to negotiate lotteries.

In this example we have assumed there are only two contracts the agents have to choose from. In general however, there may be many more contracts (recall that in a state w every legal sequence of joint moves starting at w is a contract). Since the payoff matrix of G_w would have one row and one column for every known contract it would be too time consuming to calculate the exact Nash Equilibrium of G_w for every node ν_w in the tree. Instead, suppose that for some state w we have a set of contracts $\{x_1, x_2, \dots x_m\}$ then for each pair of contracts from this set we determine what the Nash Product would be if we did allow lotteries over these two contracts, and then pick the pair for which this product is maximal. Then, we use this pair to calculate the Negotiation Values as above.

Note that every time our algorithm finds new sequences of joint moves starting at state w_l corresponding to some leaf node ν_l, the Negotiation Values of ν_l need to be updated. This is done by the function *updateNegoValuesAndResValues*() in Algorithm 1.

6.5 Reservation Values

As explained, our agent determines for each state explored in the tree a pair of Negotiation Values and in order to do so it needs to know the Reservation Values of that state. Remember that the Reservation Values are defined as those values the negotiators obtain when negotiations fail. In our case, if negotiations in the current round fail, the players still have the chance to continue negotiating in the next round. This means that if the current state is w and we know that the next state will be w^*, then the Reservation Values of state w should equal the Negotiation Values of w^*. Of course, if α_1 is the active player of w then α_0 cannot know which will be the next state. However, it can make an educated guess by assuming that α_1 is rational. If w is a non-terminal state then we define $next(w)$ as the set of all states that could be the next state:

$$next(w) = \{w' \in W \mid \exists a \in L_j(w) : w' = u(w,a)\}$$

with α_j being the active player of w. We can then predict the next state to be the state $w^* \in next(w)$ that maximizes the Negotiation Value of the active player:

$$w^* = \underset{w' \in next(w)}{\arg\max}\ nv_{w',j}$$

We can then define the Reservation Values rv_i of a non-terminal state w as $rv_{w,i} := nv_{w^*,i}$. For any terminal state the reservation values are simply defined as the exact utility values of that state: $rv_{w_T,i} := U_i(w_T)$.

In principle, we have now formally defined the Reservation Values and the Negotiation Values for any *state* w of the game, and hence for every *node* ν_w in the search tree. However, in practice, we have the problem that in order to calculate the Negotiation Values of a node ν_w we need to know its Reservation Values, and in order to calculate the Reservation Values of ν_w we need to know the Negotiation Values of the children of ν_w. This means we can only calculate them exactly if we have exhaustively generated the entire subtree under ν_w. This is a problem, because often this tree will be too large for exhaustive exploration.

Therefore, we only use the identity $rv_{\nu_w,i} = nv_{\nu_{w^*},i}$ if the children of ν_w have already been generated. If this is not the case, we can use the Average Rollout Scores $\widetilde{U}_{\nu_w,i}$ as an alternative. After all, the Average Rollout Score indicates how much a player can expect to achieve without negotiations at all, which is obviously a lower bound for the real Reservation Values. However, this approach may also fail if ν_w has not been selected often enough by the branch-selection functions, because in that case the Average Rollout Scores will not be accurate enough. In that case we simply leave the Reservation Values undefined. In summary, we define the Reservation values of a node as follows, with N_{ν_w} as defined in Section 4, and θ being some threshold value:

$$rv_{\nu_w,i} = \begin{cases} \text{undefined} & \text{if } N_{\nu_w} < \theta \\ \widetilde{U}_{\nu,i} & \text{if } N_{\nu_w} \geq \theta \text{ but } nv_{\nu_{w^*},i} \text{ is undefined} \\ nv_{\nu_{w^*},i} & \textit{otherwise} \end{cases}$$

For any node ν the Negotiation Values $nv_{\nu,i}$ are calculated according to Section 6.4, using the reservation values $rv_{\nu,i}$ if they are defined, and $nv_{\nu,i}$ is undefined if $rv_{\nu,i}$ is undefined.

PROPOSITION 1. *For any node ν, if its Reservation Values and its Negotiation Values are defined, then the Reservation Values and Negotiation Values of all its ancestors are also defined.*

PROOF. This follows from the fact that if $N_\nu \geq \theta$, then we must also have $N_{\nu'} \geq \theta$ for all ancestors ν' of ν. This holds, because a node ν is only 'selected' during the selection steps of the algorithm if its parent was also selected. □

6.6 Selecting an Action

In each round of the game, right before the deadline, the player must send a message to the server with its next action. In case the players have made an agreement (either in the current round or during one of the earlier rounds) that prescribes the agents' actions for the current round, then the agent simply chooses the action according to that agreement.

Otherwise, it needs to decide which of the currently legal moves is the best. Here again our algorithm differs from regular MCTS. While a regular MCTS picks the move that leads to the state with the highest Average Rollout Score,[5] our agent picks the move that leads to the node with the highest Negotiation Value. After all, as long as the Negotiation Values dominate the Average Rollout Scores, it is beneficial for both players to negotiate, and therefore it is reasonable to expect that negotiations will succeed.

[5]Actually, it picks the move that leads to the child node that has been selected in a branch the largest number of times, but if the tree has been expanded enough this move will be the same as the one that maximizes the Average Rollout Score.

7. EXPERIMENTS

In this section we present the experiments we have performed with our algorithm. We have tested it on the following games:

- Iterated Prisoner's Dilemma (IPD)
- The Centipede Game (CG)
- Dollar Auction (DA)

These are classical games that are often discussed in Game Theoretical text books. We should remark that none of these games were *intended* to be used with negotiations. They are usually analyzed under the assumption that the players do not communicate or make binding agreements. Therefore, one could argue that by allowing negotiations we have actually changed the rules of these games so we are in fact playing different games. Nevertheless, we still refer to our games-with-negotiations by their original names.

We feel it is important to stress here that writing a negotiating agent *specifically* for any of these games would be an easy task. However, the point of our algorithm is that it is entirely *generic*. We are using exactly the same algorithm for every game, without even changing a single parameter, and it could just as well be used for any other non-zero-sum game, as long as it is described in GDL.

Our agent does not have any knowledge about these games, other than their GDL descriptions. In particular, this means there is no straightforward, generic, way for our agent to determine their Pareto Frontiers. Furthermore, even if an agent does know which utility vectors are Pareto-optimal, this is still not enough to negotiate successfully, because it also needs to know which sequences of joint moves would lead to these Pareto-optimal outcomes.

For each of these games, we have let two instances of our agent play 100 matches against each other with negotiations and 100 matches without negotiations. When our agent plays without negotiating it just applies a plain MCTS. For each game we have given the players a deadline to negotiate of 5 seconds per round. Our algorithm is implemented in Java and the experiments were performed on a HP Z1 G2 workstation with Intel Xeon E3 4x3.3GHz CPU and 8 GB RAM. We have downloaded the GDL descriptions of the above games from the standard GDL repositories at `http://games.ggp.org/`.

The results are displayed in Table 2. Each row represents one of the games. The first column shows the outcome of the Subgame Perfect Nash Equilbrium for each game, and the fourth column shows the optimal outcome for each game. For the IPD this optimum is the utility vector for which the Nash Product is maximized. In this game not even any lottery between two utility vectors yields a higher Nash Product, so this is the obvious 'optimal' outcome. In case of the CG, there are only two Pareto-optimal utility vectors, namely $(95, 90)$ and $(80, 95)$. However, one can show that it is never rational for the first player to agree with the $(80, 95)$ outcome. We will not provide a formal proof of this, for lack of space. Informally, this is because during the first 16 rounds of the game the second player cannot threaten the first player with an early termination, because that would only result in less utility. On the other hand, if the game advances to the 17th round then the first player is already assured of at least 85 utility points even without making any agreements. In case of the DA, the Pareto Frontier consists of the utility vectors $(100, 80)$ and $(75, 100)$, but the second vector is clearly irrational, because the Subgame Perfect Equilibrium already guarantees 80 points to both players.

	N.Eq.	No Nego	Nego	*Optim.*
IPD	$(20, 20)$	$(20, 20)$	$(56 \pm 0.3, 55 \pm 0.4)$	$(60, 60)$
CG	$(5, 0)$	$(5, 0)$	$(95, 90)$	$(95, 90)$
DA	$(80, 80)$	$(80, 80)$	$(100, 80)$	$(100, 80)$

Table 2: Without negotiations the results are exactly the equilibrium outcomes. With negotiations the results are close to optimal.

The two middle columns show the average utility obtained by the two players over 100 matches, with and without negotiations respectively. We see that without negotiations, the outcome of every match in every game is exactly the theoretical equilibrium. When the players do negotiate they obtain much higher scores. In the case of the CG and the DA, the players reach exactly the optimal outcome in every match. In the case of the IPD the players score an average of 55 and 56 points respectively, with standard errors of 0.3 and 0.4 respectively, which is close to the optimal solution. The reason that the algorithm does not achieve the theoretical optimum in the IPD is simply because the IPD has a very large search space, so it does not always find the optimal contract. Each player has 2 legal actions in each round and the game lasts for 20 rounds, so there are 4^{20} possible terminal sequences starting at the initial state. For the other two games the search space is small enough to explore exhaustively.

We conclude that our algorithm enables the players to significantly increase their scores by means of negotiation, and obtain results that are very close to optimal.

The GGP repositories do contain a number of other (more complex) non-zero-sum games, such as Free-For-All, Skirmish, Chinese Checkers, and Chinook. Unfortunately, it turns out that these games are not suitable for negotiations, because even without negotiating our pure MCTS algorithm already achieves near-optimal scores.

8. FUTURE WORK

In this paper we have assumed the negotiators have full information about the game. In particular, this means that for any proposed deal, our agent was able to perfectly determine the opponent's utility value of this deal. Although we think that in real life a negotiator does have information about the opponent's utility function, it is unrealistic to assume this information is perfect. Therefore, we plan to generalize our algorithm to games described in GDL-II, which allows hidden information. The games used in our experiments were highly theoretical. It would be more interesting to see if we can implement some real-world negotiation scenarios in GDL and apply our algorithm to them. Furthermore, we would like the negotiators to be able to negotiate more complex kinds of deals, rather than just linear sequences of joint moves. They could for example make proposals of the form "I promise that if you do a then I will do b, but if you do c then I will do d". This could be realized by letting the players formulate their proposals in a recently introduced strategic language called SGL [31]. Finally, we will investigate how our algorithm can be generalized to games for more than 2 players, such as Diplomacy.

REFERENCES

[1] R. Axelrod and W. Hamilton. The evolution of cooperation. *Science*, 211(4489):1390–1396, 1981.

[2] T. Baarslag, K. Hindriks, C. M. Jonker, S. Kraus, and R. Lin. The first automated negotiating agents competition (ANAC 2010). In T. Ito, M. Zhang, V. Robu, S. Fatima, and T. Matsuo, editors, *New Trends in Agent-based Complex Automated Negotiations, Series of Studies in Computational Intelligence*. Springer-Verlag, 2010.

[3] A. Fabregues. *Facing the Challenge of Automated Negotiations with Humans*. PhD thesis, Universitat Autònoma de Barcelona, 2012.

[4] A. Fabregues and C. Sierra. Dipgame: a challenging negotiation testbed. *Engineering Applications of Artificial Intelligence*, 2011.

[5] P. Faratin, C. Sierra, and N. R. Jennings. Negotiation decision functions for autonomous agents. *Robotics and Autonomous Systems*, 24(3-4):159 – 182, 1998. Multi-Agent Rationality.

[6] P. Faratin, C. Sierra, and N. R. Jennings. Using similarity criteria to make negotiation trade-offs. In *International Conference on Multi-Agent Systems, ICMAS'00*, pages 119–126, 2000.

[7] S. Fatima, M. Wooldridge, and N. R. Jennings. An analysis of feasible solutions for multi-issue negotiation involving nonlinear utility functions. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, pages 1041–1048, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

[8] A. Ferreira, H. Lopes Cardoso, and L. Paulo Reis. Dipblue: A diplomacy agent with strategic and trust reasoning. In *7th International Conference on Agents and Artificial Intelligence (ICAART 2015)*, pages 398–405, 2015.

[9] H. Finnsson. *Simulation-Based General Game Playing*. PhD thesis, School of Computer Science, Reykjavik University, 2012.

[10] M. Genesereth, N. Love, and B. Pell. General game playing: Overview of the AAAI competition. *AI Magazine*, 26(2):62–72, 2005.

[11] T. Ito, M. Klein, and H. Hattori. A multi-issue negotiation protocol among agents with nonlinear utility functions. *Multiagent Grid Syst.*, 4:67–83, January 2008.

[12] D. de Jonge. *Negotiations over Large Agreement Spaces*. PhD thesis, Universitat Autònoma de Barcelona, 2015.

[13] D. de Jonge and C. Sierra. NB3: a multilateral negotiation algorithm for large, non-linear agreement spaces with limited time. *Autonomous Agents and Multi-Agent Systems*, 29(5):896–942, 2015.

[14] D. de Jonge and C. Sierra. GANGSTER: an automated negotiator applying genetic algorithms. In N. Fukuta, T. Ito, M. Zhang, K. Fujita, and V. Robu, editors, *Recent Advances in Agent-based Complex Automated Negotiation*, Studies in Computational Intelligence, pages 225–234. Springer International Publishing, 2016.

[15] D. E. Knuth and R. W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293 – 326, 1975.

[16] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *Proceedings of the 17th European Conference on Machine Learning*, ECML'06, pages 282–293, Berlin, Heidelberg, 2006. Springer-Verlag.

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

[18] N. Love, M. Genesereth, and T. Hinrichs. General game playing: Game description language specification. Technical Report LG-2006-01, Stanford University, Stanford, CA, 2006. http://logic.stanford.edu/reports/LG-2006-01.pdf.

[19] I. Marsa-Maestre, M. A. Lopez-Carmona, J. R. Velasco, and E. de la Hoz. Effective bidding and deal identification for negotiations in highly nonlinear scenarios. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, pages 1057–1064, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

[20] I. Marsa-Maestre, M. A. Lopez-Carmona, J. R. Velasco, T. Ito, M. Klein, and K. Fujita. Balancing utility and deal probability for auction-based negotiations in highly nonlinear utility spaces. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, pages 214–219, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[21] J. Nash. The bargaining problem. *"Econometrica"*, "18":155–162, 1950.

[22] M. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, 1994.

[23] L. Pan, X. Luo, X. Meng, C. Miao, M. He, and X. Guo. A two-stage win-win multiattribute negotiation model: Optimization and then concession. *Computational Intelligence*, 29(4):577–626, 2013.

[24] J. S. Rosenschein and G. Zlotkin. *Rules of Encounter*. The MIT Press, Cambridge, USA, 1994.

[25] R. W. Rosenthal. Games of perfect information, predatory pricing and the chain-store paradox. *Journal of Economic Theory*, 25(1):92 – 100, 1981.

[26] E. E. S. Kraus, D. Lehman. An automated diplomacy player. In D. Levy and D. Beal, editors, *Heuristic Programming in Artificial Intelligence: The 1st Computer Olympia*, pages 134–153. Ellis Horwood Limited, 1989.

[27] S. Schiffel and M. Thielscher. M.: Fluxplayer: A successful general game player. In *In: Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 1191–1196. AAAI Press, 2007.

[28] R. Serrano. bargaining. In S. N. Durlauf and L. E. Blume, editors, *The New Palgrave Dictionary of Economics*. Palgrave Macmillan, Basingstoke, 2008.

[29] M. Shubik. The dollar auction game: A paradox in noncooperative behavior and escalation. *The Journal of Conflict Resolution*, 15(1):109–111, 1971.

[30] J. von Neumann. On the theory of games of strategy. In A. Tucker and R. Luce, editors, *Contributions to the Theory of Games*, pages 13–42. Princeton University Press, 1959.

[31] D. Zhang and M. Thielscher. A logic for reasoning about game strategies. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15)*, pages 1671–1677, 2015.