Minimax Value Iterarion Applied to Robotic Soccer

Gonçalo Neto and Pedro Lima
Institute for Systems and Robotics
Av. Rovisco Pais 1, 1049-001, Lisboa, Portugal
{gneto,pal}@isr.ist.utl.pt

Abstract— **This work focuses on developing a dynamic programming algorithm to solve a class of Stochastic Games called two-person zero-sum games, inspired by the reinforcement learning algorithm Minimax-Q. In each state of the game, linear programming is used to find a Nash equilibrium, which ensures optimality in a worst-case scenario. The method is then applied to a behavioral model of a robotic soccer game. The goal is to find the worst case scenario strategy for such a team, so that a lower bound for the team's performance is guaranteed. Most of the times it converges to a conservative solution that tries, above all, to keep the opponent from scoring, rather than trying to score itself.**

Index Terms— **Stochastic Game, Dynamic Programming, Nash Equilibrium, Robotic Soccer.**

I. Introduction

The field of multi-agent robotics has been growing, in the past few years, as a promising way of solving some difficult problems otherwise hard to tackle. Particularly, robotic soccer is proving to be an interesting testbed for many algorithms and methodologies.

Several discrete event systems techniques (Cassandras and Lafortune, 1999) have been used to model such systems, not without success. For example, Markov Decision Processes (Sutton and Barto, 1998) is a framework that allows us to model decision making problems in many robotic environments. An agent living in a MDP-like world must try to maximize its reward over time by choosing appropriate actions at each state. Concepts like dynamic programming (Bertsekas, 1995; Sutton and Barto, 1998), Monte Carlo methods (Sutton and Barto, 1998) and reinforcement learning (Sutton and Barto, 1998) allow the agent to find optimal decision policies.

These techniques work very well in MDPs because they assume stationarity of the environment. On the other hand, they do not work so well for multi-agent systems where, from each agent point of view, the environment is not stationary. Game Theory (von Neumann and Morgenstern, 1947) provides a way of finding solutions for such systems by introducing equilibrium notions (Nash, 1950) and explicitly handling multiple agents and their joint actions. Particularly, Matrix Games solve the multi-agent and single state problem and Stochastic Games (Shapley, 1953) act as an extension of both MDPs and Matrix Games, working over a multi-agent and multi-state system.

Many of the concepts associated with MDPs or with Matrix Games can be extended to Stochastic Games, having produced several successful algorithms for certain subsets of Stochastic Games (Littman, 1994; Littman, 2001; Hu and Wellman, 2003; Claus and Boutilier, 1998). The work presented in this paper was inspired by that of (Littman, 1994), creating an algorithm which mixes Dynamic Programming and a Matrix Game solver to find worst-case optimal decision solutions for one of the teams, before the execution of the game. The algorithm was applied to a Stochastic Game modeling a robotic soccer environment.

The paper has the following outline: In Section II we describe the frameworks of MDPs, Matrix Games and Stochastic Games as well as some solution concepts. In Section III we present a dynamic programming algorithm – Minimax Value Iteration – to solve a particular kind of Stochastic Game. Section IV describes the Stochastic Game model used for the soccer game. Finally, Section V presents the results of applying the algorithm to the described model, Section VI discusses those results and Section VII presents future work directions.

II. Background

A. Markov Decision Processes

An MDP can be defined as a 4-tuple (S, A, T, R) where S is a set of states and A a set of available actions. $T : S \times A \times S \to [0, 1]$ is a transition function defining how the agents' actions affect the environment and $R : S \times A \times S \to \mathcal{R}$ is an expected reward function, giving some insight on the desired task for the agent.

The concept of *policy* plays a key role in MDPs in the sense that it is responsible for the decision making of the agent. Generally, a policy $\pi(s_t, a_{t-1}, s_{t-1}, a_{t-2}, ...) \in PD(A)$ (where $PD(A)$ is the set of probability distribution functions over A) is nothing more than a collection of probability distribution functions, one for each trace of the system, defining the probability that some action will be chosen. Assuming the process is Markovian (by definition for MDPs) when can just refer to $\pi(s_t) \in PD(A)$, ignoring the history of the system.

The concept of optimality in a MDP is equivalent to maximizing the expected reward, which on itself aggregates a myriad of formulations. Three possible criteria are: maximization of the expected sum of the next k rewards, maximization of the expected average reward or maximization of the sum of all future rewards (which may not converge for some MDPs), to state but a few. A usual formulation is to try to maximize, with some policy π, the discounted reward over time, with discount factor γ. In this context, we can write a value for each state defined as:

$$V^{\pi}(s) = E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s, \pi \right\}$$

We could also consider the expected reward conditioned not only by the current state but by particular action. The resulting function depends on the state-action pair and the expected rewards are usually named Q-values:

$$Q^\pi(s,a) = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s, a_t = a, \pi \right\}$$

Having defined the state values we can obtain an equation, called the Bellman equation, which recursively relates them:

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^\pi(s') \right]$$

The optimal policy is one that, for each state, maximizes the state value function $V(s)$. This will be a deterministic policy that satisfies a stronger relation, named the Bellman optimality equation:

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$

The dynamic programming algorithm known as *Value Iteration* is based on the Bellman optimality equation and can be written in a simple form:

$$V_{k+1}(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

for all $s \in S$.

It can be shown that this recursive equation assures convergence of the state values to the optimal ones (Bertsekas, 1995). The optimal policy will be deterministic, giving probability 1 to just one action, and greedy, always choosing the action that will lead to higher expected reward.

B. Matrix Games

Matrix Games are the right framework to deal with single-shot games, that is, games where there are multiple players but just one state with an associated reward structure. Formally, they can be defined by a tuple $(n, A_{1\ldots n}, R_{1\ldots n})$ where n represents the number of agents, A_i is the action set for player i ($A = A_1 \times \cdots \times A_n$ is the joint action set) and $R_i : A \to \mathcal{R}$ is the reward function of player i. One important characteristic of Matrix Games is the fact that each agent's reward function depends on the actions of all the players and not just its own actions. The name Matrix Games arises from the fact that each player's reward structure can be represented as an n-dimensional matrix.

Games like Rock-Paper-Scissors (Tables I and II) or the Prisoner's Dillema (Tables III and IV) are examples of two-person Matrix Games. One usual convention in two-person games is that the first player always specifies the row index and the second player (the opponent in adversarial situations) specifies the column index.

In this framework, the concept of *strategy* plays a similar role to that of policy in MDP. A strategy $\sigma_i \in PD(A_i)$ defines the way agent i decides on a Matrix Game. A collection of n strategies, one for each of the agents, is called a *joint strategy* and it can be written $\sigma = \langle \sigma_i, \sigma_{-i} \rangle$, where the notation σ_{-i} is used to refer to a joint strategy for all players except for player i. For every joint strategy, there is an associated reward for each of the players $R_i(\sigma)$ – note that the reward for one of the players depends on the strategies followed by all of them.

	Rock	Paper	Scissors
Rock	0	-1	1
Paper	1	0	-1
Scissors	-1	1	0

TABLE I

ROCK-PAPER-SCISSORS REWARD MATRICES – PLAYER 1

	Rock	Paper	Scissors
Rock	0	1	-1
Paper	-1	0	1
Scissors	1	-1	0

TABLE II

ROCK-PAPER-SCISSORS REWARD MATRICES – PLAYER 2

An individual strategy is said to be a *best-response strategy* if, for a given σ_{-i} played by all other players, it achieves the highest possible reward. We write:

$$\sigma_i \in BR(\sigma_{-i})$$

where $BR(\sigma_{-i})$ represents the set of all best-response strategies of player i to σ_{-i}. A *Nash equilibrium* is a collection of strategies, one for each player, that are best response strategies, which means that none of the players can do better by changing strategy, if all others continue to follow the equilibrium.

$$\forall_i \;\; \sigma_i \in BR(\sigma_{-i})$$

An important characteristic of Matrix Games is that all of them have at least one Nash equilibrium.

A usual way of classifying Matrix Games is the following:

- *Zero-sum games* are two-player games ($n = 2$) where the reward for one of the players is always symmetric to the reward of the other player. Actually, this type of games is equivalent to *constant-sum games*, where the sum of both players rewards is always constant for every joint action. An example is Rock-Paper-Scissors, as shown in Tables I and II.
- *Team-games* have a general number of players but their reward is the same for every joint action. An example is the Matrix Game shown in Tables V and VI that models a shared resource channel where the agents are playing cooperatively.
- *General-sum games* are all types of matrix games. However, the term is mainly used when the game can not be classified as a zero-sum one. An example is the Prisoner's Dillema, as shown in Tables III and IV.

The first kind of games, also called *two-person zero-sum* games, is very appealing because although they can

	Tell	Not Tell
Tell	2	0
Not Tell	5	1

TABLE III

PRISONER'S DILLEMA REWARD MATRICES – PLAYER 1

	Tell	Not Tell
Tell	2	5
Not Tell	0	1

TABLE IV

PRISONER'S DILLEMA REWARD MATRICES – PLAYER 2

	Wait	Go
Wait	0	1
Go	4	-2

TABLE V

MODELLING OF A SHARED RESOURCE – PLAYER 1

	Wait	Go
Wait	0	1
Go	4	-2

TABLE VI

MODELLING OF A SHARED RESOURCE – PLAYER 2

contain several equilibria, all of them have equal reward structure and are interchangeable. So, in this situation a Nash equilibrium corresponds to a worst-case scenario: if player 1 is playing an equilibrium strategy σ_1 then there is nothing that player 2 can do to improve its own payoff besides playing the corresponding strategy σ_2 and, because the game is zero-sum, there is no way player 1 can get a lower payoff than it is already receiving. This does not mean there are not higher reward possibilities but playing them also involves the risk of ending up receiving lower reward than in the equilibrium.

We can think of the equilibrium value as an optimal value in the sense that, for each of the players, the payoff will never be worst than that value. In alternated games the way of solving the game in the worst-case scenario is using a minimax approach where we maximize our reward given the other player is doing everything to minimize it; this procedure returns a deterministic strategy. However, in Matrix Games both players choose their actions at the same time and, in this situation, we cannot reason exactly like in alternated games.

We can still think, however, of a minimax operator that acts on the strategy space, rather than on the action space and, for the class of games considered, a way of finding an equilibrium can be written as:

$$\max_{\sigma \in PD(A)} \min_{o \in O} \sum_{a \in A} \sigma(a) R(a, o)$$

where A represents player 1 action set, O represents player 2 (the opponent) action set and $R(a, o)$ the reward when joint action $\langle a, o \rangle$ is played. The optimal value will be a probability distribution function over the agent's actions instead of a singular action. This is a linear program which can be easily solved using a simplex algorithm. For further explanation on how to formulate the problem above as a linear program refer to (Owen, 1995) or (Littman, 1994).

C. Stochastic Games

As above mentioned, Stochastic Games can be thought as an extension of Matrix Games and/or Markov Decision Processes in the sense that they deal with multiple agents in a multiple state situation. Formally, they can be defined as a tuple $(n, S, A_{1,\ldots,n}, T, R_{1,\ldots,n})$ where n represents the number of agents, S the state set, A_i the action set of agent i and $A = A_1 \times \cdots \times A_n$ the joint action set. The transition function in a Stochastic Game depends on the actions of all players $T : S \times A_1 \times \cdots \times A_n \times S \to [0, 1]$. The reward function, as T, also depends on the actions of all players $R : S \times A_1 \times \cdots \times A_n \times S \to \mathcal{R}$.

When can think of a SG as a succession of Matrix Games – a distinct game to every state. As with MDPs, the concept of policy can be defined for a Stochastic Games but, in this case, we are normally interested in the policies of each of the players $\pi_i(s) \in PD(A_i)$. Like the concept of strategy for MGs, a *joint policy* can be defined $\pi = \langle \pi_i, \pi_{-i} \rangle$ with π_{-i} denoting a collection of policies for all the players except i.

III. MINIMAX VALUE ITERATION

A two-person zero-sum Stochastic Game is one in which the Matrix Games for all of the states are two-person zero-sum ones. In this situation, we can guarantee the existence of one Nash equilibrium (or multiple interchangeable equilibria) in each of the intermediate games. So, a way of finding an optimal policy, in the Nash equilibrium sense, is to find the Nash equilibrium policy for the state values: it will guarantee that the player will not get a lower expected reward than the Nash equilibrium value. In order to adapt value iteration to make this kind of computation we should replace the max operator in the Bellman optimality equation with a minimax operator, as it was used to find equilibria for two-person zero-sum MGs. So, the Bellman optimality condition for two-person zero-sum games can be written as the following equations:

$$V^*(s) = \max_{\pi \in PD(A)} \min_{o \in O} \sum_{a \in A} \pi(a) Q(s, a, o)$$

with

$$Q(s, a, o) = \sum_{s'} R(s, a, o, s') + \gamma T(s, a, o, s') V^*(s')$$

where $a \in A$ represent the actions of one of the players and $o \in O$ the actions of the other player.

One problem with this method is that it can only be applied to two-person stochastic games with a symmetrical reward structure (hence making it zero-sum). However, it would be nice to apply it to team situations with opposite objectives, hence with a zero-sum reward structure, but with several agents. A possible approach, which we used in this work, is that of (Littman, 1994) in his Minimax-Q algorithm which thinks of teams as augmented agents and performs the minimax operation on the joint-action sets of each team, reducing the problem to a two-person zero-sum one. As for the reward question in each team, this approach assumes a purely teamwork situation, with every member of the team receiving the same reward.

So, if we have a Stochastic Game with the first n agents of one team and the remaining m agents of another team, and with the reward functions verifying $R_1 = \cdots = R_n = R$ and $R_{n+1} = \cdots = R_{n+m} = -R$, we can define a new stochastic game where:

- $A = A_1 \times \cdots \times A_n$
- $O = A_{n+1} \times \cdots \times A_{n+m}$
- $R = R_1 = \cdots = R_n = -R_{n+1} = \cdots = -R_{n+m}$

This stochastic game is now a two-person zero-sum and we can apply the Minimax Value Iteration algorithm, whose recursive equation is directly taken from the Bellman optimality equation for this kind of games:

$$V^{k+1}(s) \leftarrow \max_{\pi \in PD(A)} \min_{o \in O} \sum_{a \in A} \pi(a) Q^{k+1}(s,a,o)$$

with

$$Q^{k+1}(s,a,o) \leftarrow \sum_{s'} R(s,a,o,s') + \gamma T(s,a,o,s') V^k(s')$$

for all $s \in S$.

IV. Soccer Game Model

A. Individual Players

In this work, we chose to model by looking at two characteristics: ball possession and current player role.

The roles correspond not to specific positions in the field or to the exclusivity of doing certain actions, but rather to the opportunity of doing certain actions. The roles considered were **Attacker and Defender**. The **Attacker** role, for example, does not guarantee exclusivity of attack but, if the player shoots the ball while in the **Attacker** role, it has higher probability of succeeding in its task. On the other hand, if it tries to block an attack it will not be so successful as if it was on the **Defender** role.

As for the actions, the ones considered were:

- **get-ball -** try to get the ball.
- **shoot -** shoot for the opponent goal.
- **block -** defend own goal.
- **attack -** switch to attacker role.
- **defend -** switch to defender role.

With this approach it is also easy to define **pass** and **receive** actions but that would make an already hard computation even harder.

The finite state automaton that models such agents can be seen in Fig. 1. When can see that most actions are non-deterministic, meaning that they will not always lead to the same state. Moreover, although there is a probability structure behind those transitions, the transition probabilities depend on the states and actions of all the players (the agent, its teammates and its opponents) and so, from each agent's point of view, those probabilities are not stationary.

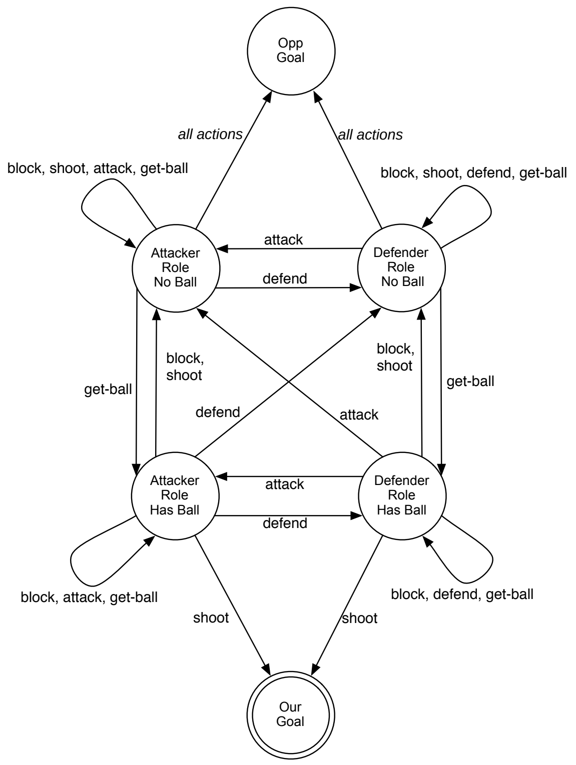

Fig. 1. Model of the players

The bottom line is: although the automaton gives an idea of what each player can do, it is usually not possible to decouple its behavior from that of the other players. The overall behavior of the stochastic game is presented in the next section.

B. Stochastic Game Definition

To construct the complete game we added to the state of each player the information about the team they belong to. The complete state space for the Stochastic Games was constructed with the combination of all individual states plus the two possible goal outcomes, with the restriction that only one player could have the ball. So, for a Stochastic Game consisiting of four players with a model like the one of Fig. 1 the total number of states would be 82. For example, the state represented in Fig. 2 is allowed. On the other hand, the one of Fig. 3 is not because two players have possession of the ball.

The transition function was defined by a set of rules summarized in Algorithm 1.

With the transition rules defined, we used a Monte-Carlo method to estimate the complete transition function. Note that, as mentioned, to apply the method we needed to consider each team as an agent, with the actions being each teams' joint actions.

The reward function was defined simply as being zero in almost every state except when we arrive at a terminal state, where there is a positive unitary reward attributed to the scoring team and a negative unitary reward attributed to their opponent, which ensures the necessary zero-sum

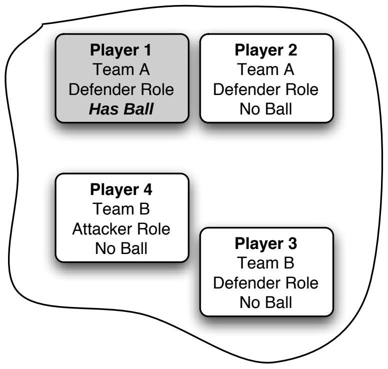

Fig. 2. One of the Stochastic Game states

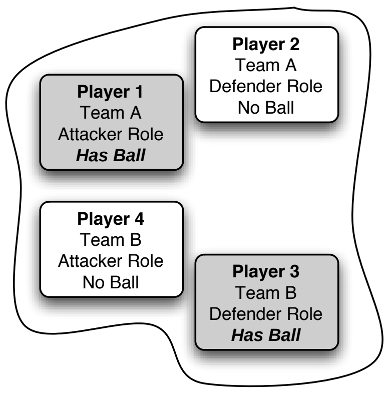

Fig. 3. A state which is not part of the Stochastic Game

condition. In fact, when the reward is backward propagated by the Minimax Value Iteration algorithm, the zero-sum condition is maintained and that is why the team only needs to maintain value-states for itself – the expected reward for the other team will always be a symmetrical value.

C. Game Setups

We tested the method with different setups using the agents presented in Table VII. Looking back at the automaton in Fig. 1 we can see that the agent of Type B corresponds to restricting the automaton to the right part while the agent of Type C corresponds to restricting the automaton to the left part. The setups considered were the

Type	Roles	Actions
A	Attacker, Defender	get-ball, shoot, block, attack, defend
B	Defender	get-ball, shoot, block
C	Attacker	get-ball, shoot, block

TABLE VII
SOCCER AGENT TYPES

ones shown in Table VIII. With each setup, we analyzed the characteristics of the Minimax Value Iteration method applied to the setup and, afterwards, tested its performance against two kinds of teams: the dual optimal team (in the Nash equilibrium sense) and a team which randomly chooses actions.

Algorithm 1 Transition Rules for Stochastic Game

```
$S \leftarrow$ State (which is a list of players)
$A \leftarrow$ Action (one action for each player)

**for all** $p \in S$ and $a \in A$ **do**
  **if** $a$ is *attack* or *defend* **then**
    change role of $p$
  **end if**
**end for**

**if** no player has ball **then**
  $n \leftarrow$ number of players with $a =$ *get-ball*
  with probability $\frac{1}{n}$ one player gets ball
**else**
  $c \leftarrow$ Carrier of the ball
  **if** $c$ has $a =$ *block* **then**
    with probability $0.5$ $c$ looses ball
  **else if** $c$ has $a =$ *shoot* **then**
    $t \leftarrow$ team of $c$
    $c$ looses ball
    **if** $c$ has *Attacker* role **then**
      $attackFactor \leftarrow 0.8$
    **else**
      $attackFactor \leftarrow 0.2$
    **end if**
    $k \leftarrow$ number of Attackers not of $t$ doing *block*
    $m \leftarrow$ number of Defenders not of $t$ doing *block*
    $defenseFactor \leftarrow 0.4 \times m + 0.1 \times k$
    **if** $defenseFactor < attackFactor$ **then**
      Team $t$ scores a GOAL
    **end if**
  **else**
    Do nothing
  **end if**
**end if**
```

Setup	Our Team	Opponent Team
Setup 1	1 type A + 1 type C	1 type A + 1 type C
Setup 2	1 type A + 1 type B	1 type A + 1 type B

TABLE VIII
TEAM SETUPS

V. RESULTS

A. Optimal Computation

The first setup described in Table VIII had the following characteristics: $\#S = 22$ states and $\#A = \#O = 15$ actions. So, a total number of 22 state-values and 4950 Q-values had to be stored. Nevertheless, although these numbers are not very high, it still took approximately 4706 seconds to complete 250 steps in a 865 Mhz PowerPC G4 machine, with 640Mb of RAM. Note that each step performs one update on all state-values and Q-values, corresponding to all states in S, and solves a total of 5500 linear programs.

However, the convergence rate was high and, as can be seen in Fig. 4, usually after 5 iterations the difference between consecutive values was smaller than $1x10^{-4}$. For the second setup, the number of states and actions is exactly the same and, in this case, so is the size of the state-values table and Q-values table. However, the same number

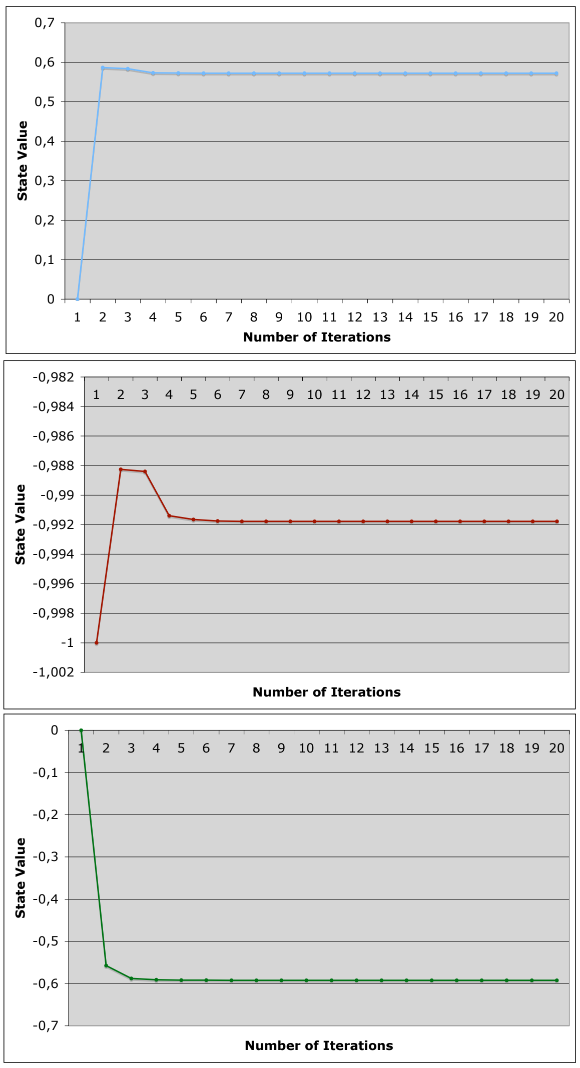

Fig. 4. Convergence of Minimax Value Iteration for three different states

of steps of the Minimax Value Iteration algorithm took 3418 seconds, with the same machine, a bit faster than the previous case. This is probably due to a higher sparsity of the constraint matrices making it easier for the simplex to compute the equilibrium policies.

The convergence in this situation was faster because the equilibrium policies converged to defensive tactics, due to the fact that one of the players could not attack very well (the action **shoot** in the **Defend** role is not very successful). In fact, the optimal policy in this setup was deterministic for almost all states, favoring actions like **defend** and **block** over more aggressive actions. The optimal strategy seemed to keep both players just defending – not scoring goals but not suffering any either.

B. *Game Simulation*

In terms of simulation, Table IX summarizes the results from both cases. Surprisingly, in most of the cases none of the teams scored any goal, after 10000 steps of the simulation. For Setup 1 playing against its dual, both teams

Test	Goals Scored	Goals Suffered
Optimal Setup 1 vs Random	2974	326
Optimal Setup 1 vs Dual Optimal	0	0
Optimal Setup 2 vs Random	0	0
Optimal Setup 2 vs Dual Optimal	0	0

TABLE IX
RESULTS AFTER 10 000 SIMULATION STEPS

have Nash equilibrium policies and they do not allow each other to reach a state where one of them has the advantage. In fact, with complete identical teams (but with opposite objectives) it would be expected that none of them could exploit the other as both play the equilibrium. Note that, when playing against a random opponent, the Nash team had quite a big advantage, winning 90% of the games played.

In setup 2 not even playing against a random opponent the team scored a goal. As referred above, the optimal policy takes advantage of the fact that there is always a defender in the field. So, while training the method assumes the worst possible outcome (according to the minimax principle) and just tries to defend, doing it perfectly – it is so focused on defending that seldom tries to get the ball.

VI. CONCLUSIONS

From all types of multi-robot systems, competitive and adversarial ones have some particular interest due to the compromises each team as to learn. Robotic soccer is a great example: each team has to attack while maintaining a solid defense base. By using the Minimax Value Iteration method, we were able to find an policy that follows a notion of equilibrium – the Nash equilibrium – in each of the states, based on the assumption that the other player will always play the policy that could damage our team the most (considering the objectives of both teams are symmetric).

Our tests showed that with a team that favored attack, the Nash equilibrium finder converged to a policy that did its best to defend itself, at least against another optimal team, but still risked a bit, which gave it advantage against a random player. In a defensive setup, the algorithm converges to a purely defensive tactic, preferring to keep its score rather than trying to achieve higher rewards. So, we can say that a equilibrium player is never a risky one and, in most situations, it could better serve as a starting point for future online learning algorithms, which would take an adaptation approach and improve the performance to a better one, exploiting the opponents weaknesses.

As for the method itself, it was shown that even for a small number of states, it still runs very slowly if the associated linear programs have large constraint matrices (the same that define the Matrix Game for each state), converging relatively fast nevertheless in terms of total number of steps. This problem is worst when the joint-space-set for each of the teams is considerably larger. Still, if we have a estimate of the game model (opponents, reward and transitions), the Dynamic Programming algorithm presented can be applied before any real game and provide an equilibrium policy from where best-response learners, which try to adapt to the current opponent team rather than to the Nash team with same joint-action set, could make adjustments.

In fact, we see our algorithm as a way of providing a knowledge base, a starting point for the online reinforcement learning algorithms that would be applied during the game itself, working as a kick-off and lower bound to what these adaptation methods can do - if the team is in a losing stream it can always return to the equilibrium policy and try to maintain the result in hands.

Another limitation of the method is the fact that it assumes full observability, in the state and in the opponent actions. Again, after obtained the equilibrium policy we just need to know the state we are in to obtain the policy. This fact works in favor of the application of Minimax Value Iteration prior to the game itself, assuming we have an estimated model of the game. As for the state observability problem, it would be interesting to study what happens if we assume full-observability while running the algorithm but partial observability during the simulation itself.

VII. Future Work

Picking up where the previous section left, an interesting line of work for future development is to study the observability problem, whether training with full observability and playing with partial observability or always using partial observability. Following the hypothesis, our interest is to use the method as prior training of policy reinforcement learners and, in this situation, the model for training could assume full-observability, provide we have sufficient information to build such a model. Nevertheless, it would always be interesting to test both situations.

Another important question is the fact that the reinforcement learning / dynamic programming algorithm has to change according to whether we are playing against an opponent or trying to do teamwork, as shown by the work of (Littman, 2001) – he uses a different algorithm depending on the the other player being a friend or a foe. Perhaps even between opponents, the algorithms or parameters should be different, according to the aggressiveness of the opponent . A possible future work direction is to create some measure of to what degree each agent ”likes” another agent and affect the global learning / dynamic programming algorithm of such measure. This implies that the agent is able to distinct between other agents which, itself, is surely another topic for discussion.

As a final proposal of future work directions, we point the fact that the training would be much faster if some actions, which we know do not make sense, were disabled in some states – this would produce much smaller linear programs and a much faster algorithm. An example is the fact that actions **shoot** and **pass** produce no change in the outcome if the player does not have the ball. To accomplish this, we could use supervisory control to disable actions that lead us to dangerous states or leading us nowhere – note that in this situation we are only interested in the logic aspect of the process. An example is the theory described in (Cassandras and Lafortune, 1999) but something like a rule-based system could also be used – it is a question of how to implement the supervisor.

References

Bertsekas, Dimitri P. (1995). *Dynamic Programming and Optimal Control*. Vol. 1. 2nd ed.. Athena Scientific.

Cassandras, Christos G. and Stéphane Lafortune (1999). *Introduction to Discrete Event Systems*. Kluwer Academic Publishers. Boston.

Claus, Caroline and Craig Boutilier (1998). The dynamics of reinforcement learning in cooperative multiagent systems. In: *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. AAAI Press.

Hu, Junling and Michael P. Wellman (2003). Nash Q-learning for general-sum stochastic games. *Journal of Machine Learning Research* **4**, 1039–1069.

Littman, Michael L. (1994). Markov games as a framework for multi-agent reinforcement learning. In: *Proceedings of the Thirteenth International Conference on Machine Learning*. New Brunswick. pp. 157–163.

Littman, Michael L. (2001). Friend-or-foe Q-learning in general-sum games. In: *Proceedings of the Eighteenth International Conference on Machine Learning*. Williamstown. pp. 322–328.

Nash, John F. (1950). Equilibrium points in n-person games. In: *Classics in Game Theory*. Princeton University Press.

Owen, Guillermo (1995). *Game Theory*. 3rd ed.. Academic Press.

Shapley, L. S. (1953). Stochastic games. In: *Classics in Game Theory*. Princeton University Press.

Sutton, Richard S. and Andrew G. Barto (1998). *Reinforcement Learning*. MIT Press.

von Neumann, John and Oskar Morgenstern (1947). *Theory of Games and Economic Behavior*. Princeton University Press. Princeton.