

A Real-Time Proxy for Flexible Teamwork in Dynamic Environments

Ivan M. Monteiro and Luis Otavio Alvares
Instituto de Informática, UFRGS, Brazil

Abstract

The demand for believable behavior in computer games has driven the research in artificial intelligence to improve character's behavior in games. However, little has been done to improve the collective behavior in partially observable, dynamic, and stochastic environments. In this work, teams of agents are developed, based on *Joint Intentions*, for environments with such features as the game Unreal Tournament 2004. Because of some limitations of existing tools, we introduce a new proxy-based tool to help agents to be a teammate. Experiments have shown that it is feasible to use a distributed approach and the advantages of our tool face to *Machinetta*.

Keywords:: Teamwork, Multi-agent System, Artificial Intelligence, Distributed System

Author's Contact:

{imonteiro,alvares}@inf.ufrgs.br

1 Introduction

In computer games, the demand for believable behavior has grown, in order to decrease the distance to the human behavior [de Byl 2004]. However, the importance given to behavioral aspects is usually less than the importance given to graphic aspects. When the subject is team-based games the situation is worst, because bots do not assume coherent roles inside the team. Thus, although there is progress on individual behavior, little has been done for social behavior.

Some team-based games, such as sport games, seem to use teamwork¹, but in reality they use a centralized solution that access the full knowledge about the state of the entities. Thus, a soccer game does not need a multi-agent coordination to perform a team move, because all bots have a centralized control. This centralized solution also allows bots to cheat the human, using knowledge about hidden state. It makes the game development easier but the result are games less believable.

The immediate advantages for distributed control of bots are the new possibility of gameplay and the load balance of bots computation. The intuition leads to belief that with more computation power it becomes easier to develop more believable behavior. This model is interesting for massive multiplayer games, that are inherently distributed. Indeed, it opens new possibilities for other kinds of games.

Teamwork has emerged as the paradigm for coordinating cooperative agents in dynamic environments and it has been shown to be capable of leading to flexible and robust behavior [Sycara and Sukthankar 2006]. It has been applied to many domains, such as: rescue disaster [Nair et al. 2000], militar combat [Hill et al. 1997] [Jones et al. 1999], robocup soccer [Kaminka 1999] [Marsella et al. 1999] [Marsella et al. 2001], and collaboration between human and agents [Scerri et al. 2003c]. Flexible teamwork, promoted by the explicit team model, that defines commitments and responsibilities for teammate, allows a robust coordination, keeping coherence even with individuals faults and unpredictable changes in the environment [Steven 2005].

Some theories have been proposed to formalize the behavior of a group of agents to act as a team. Two of these theories have important results for real problems. They are *Joint Intentions* [Levesque et al. 1990] and *Shared Plans* [Grosz and Kraus 1996]. Both are described through logic formalism, but with different approaches. *Joint Intentions* focus on a team's joint mental state, while *Shared*

Plans focus on the agent's intentions towards its collaborator's action or towards a group's joint action.

Tools as *STEAM* [Tambe 1997] and *Machinetta* [Steven 2005] had successful in many complex environments [Hill et al. 1997] [Tambe and Zhang 1998] [Tambe and Zhang 2000] [Marsella et al. 2001] [Jones et al. 1999] [Pynadath and Tambe 2003] [Scerri et al. 2003a] [Schurr et al. 2005] [Scerri et al. 2003c] [Scerri et al. 2004] [Chalupsky et al. 2001] [Scerri et al. 2001] [Scerri et al. 2003b] [Yen et al. 2001]. The natural way would be to use some of these tools to improve teamwork on bot's development in computer games. However, practical issues have pointed out for the development of a new specialized tool. *STEAM* was developed using an old version of *SOAR* [Laird et al. 1987], that made changes in its language since than. *Machinetta*, that is a successor of *STEAM*, has shown many limitations for the game domain.

The aim of this work is to promote social behavior for agents inside computers games. The paradigm known in multi-agent systems as teamwork is used to achieve this goal. In this context, a team is a group of autonomous pro-active entities, with capability of reflection about their own abilities and the abilities of the group that share a common goal.

This work also introduces *TWProxy*, a new tool that enables agents to perform teamwork. *TWProxy* uses many of the good ideas from *Machinetta*, avoiding some of its limitations and adding new features. Thus, this article focuses on the development of *TWProxy* and its usage on the development of team bots. Tests and evaluation compare how good this new solution is.

The remainder of the paper is organized as follows: Section 2 shows the main related work, Section 3 describes the environment provided by the game, Section 4 describes the features of *TWProxy*, Section 5 presents tests and evaluation, and Section 6 concludes the paper.

2 Related Work

In a teamwork problem, a group of agents acts coordinated, in a cooperative way, to achieve a shared goal. The main difference between classical coordination and teamwork is how to achieve a shared goal. Classical coordination uses a set of pre-defined plans to apply according to the situation. The teamwork extends the classical coordination using cooperation and commitment between agents, been applicable in other domains that are not the *locker-room agreements* [Stone and Veloso 1998].

Theoretical works in agent teamwork [Levesque et al. 1990] [Grosz and Kraus 1996] [Grosz and Kraus 1998] define some features about team behavior. They describe what the agents need to share in order to perform a teamwork: goals to achieve, plans to perform together, and knowledge about the environment. The agents need to share their intentions to perform a plan in order to achieve a common goal, the teammates need to be aware about their capabilities and how to fill roles to perform the high level team plan. They must be able to monitor their own progress and the group progress toward team goals. Many systems have been developed using the teamwork idea as team with human collaboration [Scerri et al. 2003a], teams for disaster situation [Nair et al. 2000], and teams for industry production line [Jennings 1995].

The formalism about *Joint Intentions* [Levesque et al. 1990] focuses on teammates' joint mental states. A team jointly intends a team action if team members are jointly committed to completing such team action, while mutually believing that they were doing it [Tambe 1997]. *Joint Intentions* assume that the agents are modeled in a dynamic multi-agent environment, without complete or correct beliefs about the world or other agents. They have changeable goals and their actions may fail. Thus, the teammate jointly commits to

¹ A cooperative effort realized for teammate to achieve a goal.

make public the knowledge if a goal becomes achieved, impossible to be reached, or irrelevant.

In contrast with *Joint Intentions*, the concept of *SharedPlans* [Grosz and Kraus 1996] relies on a novel intentional attitude, *intending-that*. An individual agent's *intending-that* is directed toward its collaborator's action or toward a group's joint action. It is defined via a set of axioms that guide an individual to take actions, including communication actions, that either enable or facilitate its teammates, subteam or team to perform the assigned tasks [Tambe 1997].

Based on these theories, important teamwork infrastructures have been developed, like *STEAM* [Tambe 1997] and *Machinetta* [Steven 2005]. *STEAM* was developed to provide flexibility and reusability of agent coordination through the use of a general model of teamwork. Such model exploits the autonomous reasoning about coordination and communication. Previous works had failed in fulfilling responsibilities or in discovering unexpected opportunities, once they had used precomputed coordination plans, which are inflexible. Indeed, they had been developed for a specific domain, what hampers the reuse. *STEAM* is specially based on *Joint Intentions* in order to make the basic building blocks of teamwork, and it uses *Shared Plans* to avoid an agent undo the actions of another.

Machinetta is a proxy-based integration infrastructure where there is a beneficial symbiotic relationship between the proxies and the team members. The proxies provide teamwork models reusable for heterogeneous entities. *Machinetta* provides each agent with a proxy and the proxies act together in a team. The *Machinetta* project was build over previous works like *STEAM* and *TEAM-CORE proxies* [Pynadath and Tambe 2003]. The pair agent-proxy execute Team-Oriented Programs(TOP) [Scerri et al. 2004], abstract team plans that provide high-level description of the activities to be performed. Such programs specify joint plans of the team, but do not contain all of the required details of coordination and communication. The team plans are reactively instantiated and they allocate roles to capable agents.

3 The Environment

In this work, the case study is *Unreal Tournament 2004 (UT2004)*, a multiplayer *First-Person Shooter(FPS)* game. The classical team-based game type *Capture The Flag* was chosen to evaluate the team coherence, because it is easy to measure the results and simple to identify the team behavior. The *GameBOTs* [Andrew Scholer 2000] [Kaminka et al. 2002] project is used to communicate the developed bot with the game.

The *GameBOTs* project works as an extended game control that enables the character to be controlled via socket TCP. Thus, it is possible to develop agents to control characters using any language with socket support. This modification offers four new gametypes, *Remote Team DeathMatch*, *Remote Capture The Flag*, *Remote Double Domination* and *Remote Bombing Run*, that can be used as environment to test bots. Using the remote connection, the bot has perception about the environment, similar to the human player. Figure 1 shows the way points seen by the bots.

The UT2004 game provides an environment with the following features [Russel and Norving 2004]:

- **partially observable** - the agent, using its perception, can not access all relevant environment states;
- **stochastic** - the next state of the environment is not completely determined by current state and actions selected by the agents;
- **nonepisodic** - the quality of bot's actions depends on its previous actions;
- **dynamic** - the environment state can change between the agent's perception and agent's actions;
- **continuous** - the number of environment states is unlimited;
- **multi-agent** - there is cooperative and competitive interaction between agents.

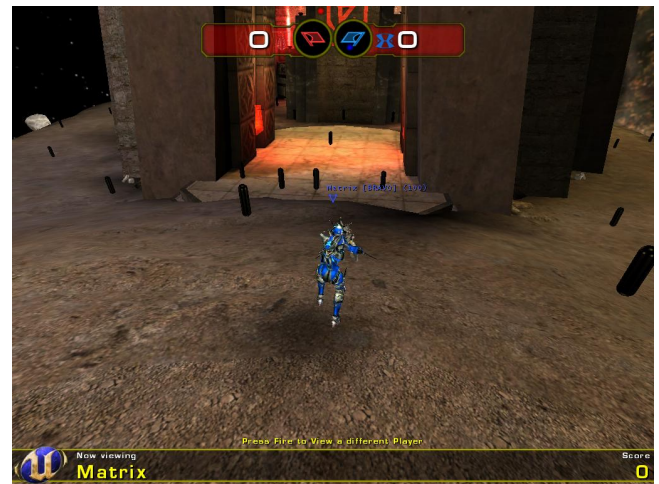


Figure 1: Waypoints seen by the bot.

When the bot is connected to the game, sensory information is received through the socket connection and the agent can act by sending commands to the game. These commands define the character behavior inside the virtual world, controlling moves, attack, and communication with other characters. The interaction between an agent and UT2004 is realized through messages exchange. The perception messages may be synchronous or asynchronous, and their complete description can be found in [Andrew Scholer 2000].

The synchronous messages arrive in batch, with a configurable interval. They are formed by visual information and the state of the agent. The asynchronous messages indicate events occurred in the environment. These events may be a damage taken, hit an obstacle, broadcast message from another agent, and other information about the game.

Actions performed by the characters are results of commands sent by the agent. An important issue about this is that the commands have persistent effects, it means that a move command will execute until the bot achieves a point or until it receives a new command that conflict with the previous, like move in another direction. Thus, the command scheme is near to human interface, where the human player need release a mouse button to stop the attack.

4 TWProxy

In order to meet the requirements for the development of teams of agents in the games domain, this work introduces the *TWProxy*, a new lightweight and efficient infrastructure to provide coordination to teams of heterogeneous agents. The *TWProxy* is based on the *Joint Intentions* [Levesque et al. 1990] formalism and it is inspired on *Machinetta*, adding new important features, including: (i) a simple and extensible language to describe plans and beliefs, (ii) atomic communication between agents, (iii) reuse of terminated plans, and (iv) plans that are invariant to the number of agents.

In our approach, summarized in Figure 2, each agent is related to a proxy that performs the communication and the multi-agent planning. The team coordination is achieved using role allocation. Each proxy knows about the capabilities of other teammates and when plan preconditions are achieved, and roles are assigned according to the current plan. When the plan postconditions are achieved, the teammate can release the role. The proxy does not tell how to execute the role, it just deliberates about what to do. Because of the high-level coordination, a flexible teamwork is possible, leaving the details about role execution with the agent.

The model about role allocation is similar to the common teamwork in the real world. When people need to execute some action together, they usually divide the task in subtasks that each one can perform. In the real world, people also use the role allocation to coordinate actions. Thus, the responsibility is distributed, and if there exists commitment between the teammates, the entire group

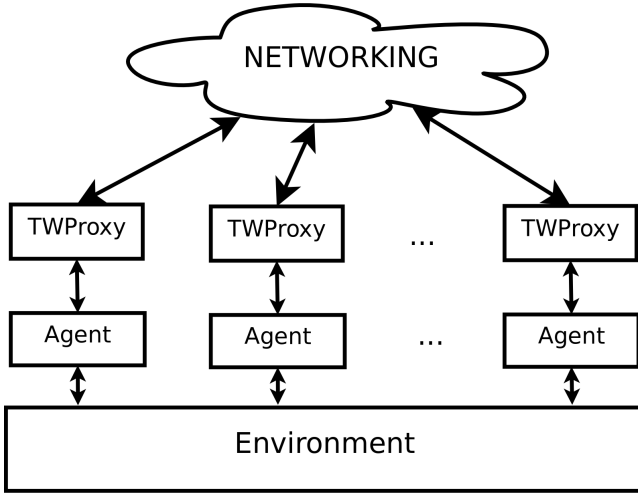


Figure 2: Teamwork model based on proxy.

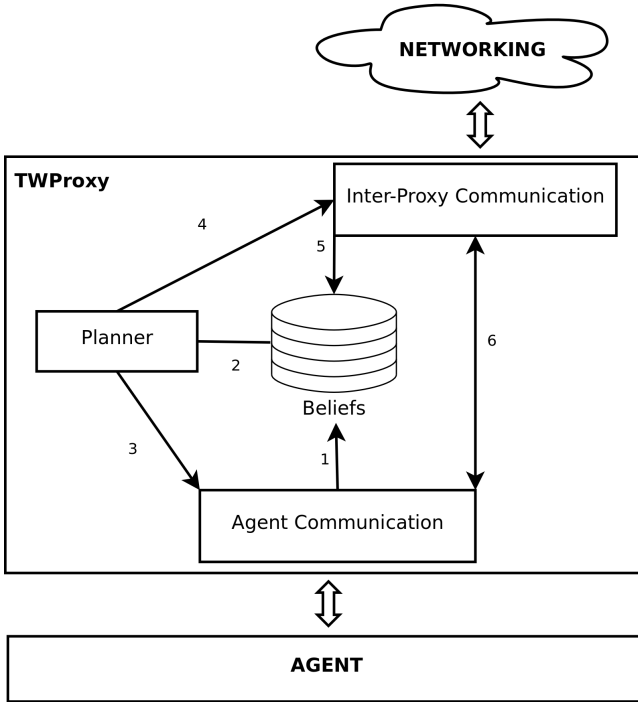


Figure 3: TWProxy organization.

acts consistently.

TWProxy also provides the functionality of distributed blackboard because it keeps the shared beliefs consistent. This belief repository helps the agents to take a decision. Due to the environment to be partially observable, the agents have sensory restrictions. Therefore, the *TWProxy* can store some shared information that is not directly sensed by the agent.

Figure 3 shows the *TWProxy* internal organization. Link 1 indicates that information from the agent can update the knowledge base. The set of beliefs can be used by the planner (link 2) that can deliberate about role allocation (link 3 and 4). Information from other proxies can update the set of beliefs (link 5) and an agent can share information using the *TWProxy* (link 6).

4.1 Describing Plan and Beliefs.

The initial agent beliefs and its plans are stored in a structured file described with the language shown in listing 1. This file contains agent beliefs about: capabilities, team composition and the specific domain. It also describes the roles and the high-level team plan.

Teams of Agent-proxy pairs execute Team-Oriented Programs (TOPs) [Scerri et al. 2004]. These TOPs specify the joint plans of the team, and the inter-dependencies between those plans, but do not contain all the required details of coordination and communication.

Listing 1: Grammar of the description language

```

<bif> ::= <block> |
        <block> <blf>

<block> ::= "belief" <belief_block> |
            "role" <role_block> |
            "plan" <plan_block>

<belief_block> ::= <id> "{" <inside_belief> "}"

<inside_belief> ::= <attr_value> |
                  <attr_value> <inside_belief>

<attr_value> ::= <id> ":" <value> ";"

<value> ::= <id> |
            <id> "," <values> |
            <number> |
            <number> "," <value> |
            "true" |
            "false"

<role_block> ::= <id> "{" <inside_role> "}"

<inside_role> ::= <attr_value> |
                 <attr_value> <inside_role>

<plan_block> ::= <id> "{" <inside_plan> "}"

<inside_plan> ::= "roles" ":" <role_values> ";"
                 <pre_block> <post_block>

<role_values> ::= <id> |
                 <id> "," <role_values> |
                 <id> "+" |
                 <id> "+" <role_values>

<pre_block> ::= "precondition" "{" <expr> "}"

<post_block> ::= "postcondition" "{" <expr> "}"

<expr> ::= "(" <expr> ")" |
          <id> "." <id> <comp> <number> |
          <id> "." <id> <comp> <id> "." <id> |
          <id> "." <id> <comp> "true" |
          <id> "." <id> <comp> "false" |
          <expr> "|" <expr> |
          <expr> "&" <expr>

<id> ::= "[a-zA-Z][a-zA-Z_0-9]*"

<number> ::= "[+-]?[0-9]+| [+]?[0-9]+\.[0-9]+"

<comp> ::= "==" | "!=" | "<" | "<=" | ">" | ">="

```

Listing 2,3,4,5 and 6 show examples of different parts of the beliefs file for the *Capture The Flag* domain. Listing 2 defines that the agent *agent001* can perform the role *CTFProtectTheBase* if it is not executing another role.

Listing 2: An example about capability belief

```

belief capability_agent001_CTFProtectTheBase {
  type: capability;
  rapId: agent001;
  load: 100;
  roleId: CTFProtectTheBase;
}

```

Listing 3 shows the belief about the team composition, and in this case also describes that agents should communicate with a peer to peer connection.

Listing 3: An example about team composition

```

belief teamComposition {
  type: teamComposition;
  members: agent001, agent002, agent003;
  myself: agent001;
  agent001_host: localhost;
  agent002_host: localhost;
  agent003_host: localhost;
}

```

```

agent001_port1: 6001;
agent002_port1: 6002;
agent003_port1: 6003;
agent001_port2: 7001;
agent002_port2: 7002;
agent003_port2: 7003;
}

```

The domain specific belief is also described in the beliefs file, as shown in listing 4. The belief contents is flexible to describe new properties, because the new attributes are handled dynamically.

Listing 4: An example about domain specific belief

```

belief flag_enemy{
    type: flag;
    owner: enemy;
    have: false;
}

belief flag_friend{
    type: flag;
    owner: friend;
    have: true;
}

```

It is possible to define meta-information about role allocation. In the example of listing 5, *priority* helps to solve allocation conflicts, *before* defines actions to execute before allocation, and *preference* defines who should receive this role. These parameters of the meta-information should be handled through inheritances of base classes in the TWProxy, otherwise, just *priority* has effect.

Listing 5: A role example

```

role CTFRecoverMyFlag {
    priority: 3;
    before: getAllRapPosition;
    preference: nearToTheEnemyBase;
}

```

Listing 6 shows an example of a plan for the *Capture The Flag* context. When a team has its flag and it does not have the enemy flag, plan *p1* is activated launching a role allocation process. This plan specifies two roles to be allocated, *CTFProtectTheBase* and *CTFCaptureTheFlag*, where the second is flexible for more than one allocation, it is indicated by signal + in the end of the role's name. If a team has four free agents, one performs *CTFProtectTheBase*, while three execute *CTFCaptureTheFlag*.

Listing 6: A plan example

```

plan p1{
    roles: CTFProtectTheBase, CTFCaptureTheFlag+;

    precondition{
        ( flag_friend.have == true ) &
        ( flag_enemy.have == false )
    }

    postcondition {
        ( flag_friend.have == false ) |
        ( flag_enemy.have == true )
    }
}

```

4.2 Communication

The inter-proxy communication assumes the usage of atomic communication over a network without shared memory. To solve the problem of atomic communication, *TWProxy* was developed over a simple atomic broadcast layer. However, it is possible to extend base classes to develop a new atomic broadcast layer. Thus, the sharing beliefs is done by atomic broadcast [Défago et al. 2004], and just the specific order messages (like to assign a role) is performed via unicast.

Two communication channels are used by each instance of TWProxy. The first channel is used to share knowledge and the second is used to exchange information about role allocation. The messages sent through these channels are serialized objects that are recomposed by a factory in the other side.

4.3 Planner

The *TWProxy* planner uses team leader to launch a new plan. The first team leader is defined statically but it may be changed dynamically at run-time. The usage of team leader avoids to solve conflict problems in role allocation, saving time to react to the environment changes. The team leader has updated beliefs and it can launch a plan consistently, because everyone is committed to share relevant information.

The planner is like a rule-based system. Every plan contains rules to activate it (the preconditions) and to stop it (the postcondition). These rules are checked using the current knowledge. Roles are assigned in the activation of a plan, and in the stopping, the involved agents can release their roles.

4.4 Agent Interface

Each individual agent is attached to a *TWProxy* forming a social agent. The communication between the individual agent and the *TWProxy* is defined by a flexible interface, allowing interaction with several kind of agents.

The interface between proxy and agent is defined by a base class that must be extended to implement this communication. Such interface is based on message exchange and it can be adjusted to each kind of agent. In a new domain, the interface with the agent is the main modification needed. Thus, it is possible to make a team, to a new domain, developing just the interface between agent and proxy, and the team program. The individual agent does not need to have social commitment, because the *TWProxy* does this for it.

5 Experiments and Evaluation

In our experiments, the agents have been developed using the framework IAF [Monteiro and dos Santos 2007], that implements a hybrid agent architecture [Monteiro 2005]. Such agents communicate with the game controlling a player. For these experiments every agent has the same capabilities. Thus, just the teamwork is different. In order to compare the *TWProxy* with another proxy, *Machinetta* is used, because of large number of successful applications [Scerri et al. 2003a] [Schurr et al. 2005] [Scerri et al. 2003c] [Scerri et al. 2004] [Chalupsky et al. 2001] [Scerri et al. 2001] [Scerri et al. 2003b].

A great concern in the teamwork problem is to avoid teamwork inconsistency. In the *Capture The Flag* domain, a teamwork inconsistency may be the following: Given a team composed by agents *A*, *B* and *C*, if either *A* or *B* are trying to recover a flag that was recovered by *C*, then the team is in an inconsistent state. Every team in a dynamic environment will stay in a inconsistent state during some period of time. The goal is to minimize inconsistency states.

The first step to develop a team is the modeling of the problem in the multi-agent context. It is necessary to divide the team actions in roles that will be performed by the agents. After that, it is possible to develop, in the agent, the abilities to execute the roles. The next step is the definition of knowledge and plans under *Team-Oriented Programs*.

This work shows two kinds of experiments. The first measures the time of teamwork inconsistency, evaluating the elapsed time between an important change in the environment and the team adoption of a new strategy. The second kind of experiment are matches that intend to show teamwork efficiency using the *TWProxy*, in order to show that it is possible to use the multi-agent paradigm and distributed artificial intelligence in the domain of modern games.

In the experiment of the time of teamwork inconsistency, agents send a sequence of changes about the environment, and the time of team reaction using *TWProxy* and *Machinetta* is calculated. The battle experiments play *Capture The Flag* where one team uses *TWProxy* and the other team uses *Machinetta*, self game BOTS or human players.

In the battle, the game UT2004 is used to provide the *Capture The Flag* gameplay. Thus, it is possible to evaluate the team actions,

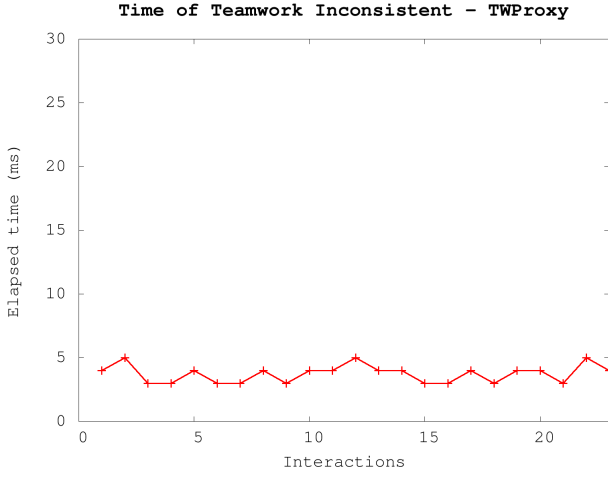


Figure 4: Time of Teamwork Inconsistency - TWProxy.

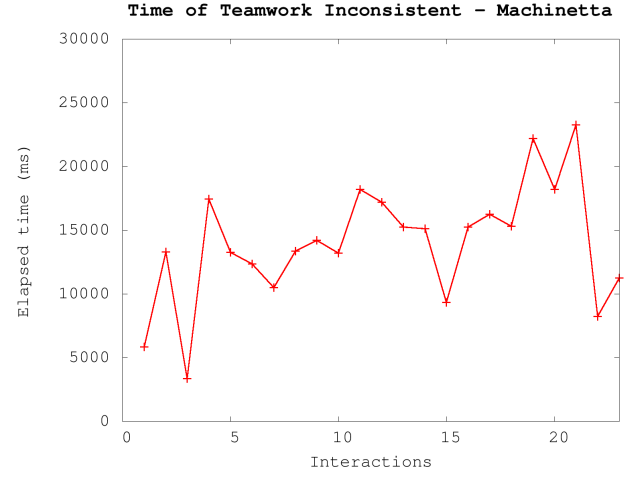


Figure 5: Time of Teamwork Inconsistency - Machinetta.

because it is easy to see which role the agent is performing, and it is also possible to measure the teamwork efficiency using the game score.

The team, that plays *Capture The Flag* using the *TWProxy*, has just four plans, that are activated when the state of a flag changes. Such plans generate the following team behavior:

1. when a team has its own flag but it has not the enemy's flag, someone needs to protect the base while others go to capture the enemy's flag;
2. when a team has both, its own and the enemy's flag, all agents must go back to their base;
3. when a team does not have its own flag but it has the enemy's flag, the agent that has the enemy's flag must go back to its base and the others should recover the team's flag;
4. when the team has no flags, every agent must search any flag.

The *Machinetta* proxy showed some limitations in the experiments, and a critical problem for this context was the impossibility to reuse a terminated plan. Thus, it was necessary to create many redundant plans to keep the bots playing the game. Other problem was the memory and the CPU usage, being impracticable to apply in a real game. However, *Machinetta* has many successful applications, being interest for comparison purposes, as will be shown in the next section.

5.1 Evaluating Time of Teamwork Inconsistency

The experiment about time of Teamwork Inconsistency (TTI) intended to evaluate the ability of *TWProxy* to react in (soft) real-time to environment changes. In order to compare its performance, *Machinetta* was exposed to the same situation. A group of three agents was updating their beliefs about the flag status, and an external entity was listening the communication between agents and proxies. Therefore, this external entity was computing the time between the environment change and the role allocation to the whole team.

The sequence of environment changes was the same for *Machinetta* and *TWProxy*. It was composed by 23 updates and it represented a whole match. Each proxy played 40 matches, in order to get a general behavior of the TTI. This experiment was performed in a single machine. For this reason, the network latency did not appear in the results.

The results shown in Table 1 describe the mean (μ) and the standard deviation (σ) of the experiments set. The measure unit is milliseconds and the time represents the total time between a new belief received and the modification of the strategy. *TWProxy*, in this experiment, achieved a mean of 3.9ms, while *Machinetta* got 12182.40ms. In other words, *TWProxy* was more than three thousand times faster than *Machinetta* to reach a consistent state. The

Table 1: Mean and standart deviation of experiment.

-	Elapsed time(ms)	
-	μ	σ
TWProxy	3.9	0.79
Machinetta	12182.40	7159.01

standard deviation shows how stable *TWProxy* execution was comparing with *Machinetta*.

Table 2 presents an example of values of TTI for one match. In this table it is possible to see the difference of magnitude between *TWProxy* and *Machinetta*. Figures 4 and 5 show the same example of Table 2 in a graphical format, in order to show the stability of the *TWProxy* in this application. It is important to say that these graphics are in different scales in an effort to compare the data dispersion.

The performance and stability of *TWProxy* against *Machinetta* was the main result from this experiment. In the midst of this circumstance, *Machinetta* also presented a high usage of memory (more than 100MB per proxy), making it difficult to use in modern games. *TWProxy*, in these experiments, did not use more than 1.6MB of memory. The key of these results relies on the kind of applications for which *Machinetta* was designed. While *TWProxy* design was concerned in achieve real-time requirements, *Machinetta* was originally developed to allow humans to participate in teams with artificial agents. Several points may explain this efficiency, including: the usage of C++ instead of Java; optimization in access to the beliefs; and mainly the fewest conflicts to resolve, because the communication with atomic broadcast guarantees the total order of message and the usage of team leader avoids the role allocation conflict.

5.2 Teamwork Efficiency

The second kind of experiment evaluates the teamwork efficiency using the gameplay *Capture The Flag*. Each experiment spent 15 minutes and it was played in order to assess different aspects. The first experiment evaluates *TWProxy*'s efficiency against *Machinetta* using the same agents in a battle, but with different proxies. The second experiment was composed by battles between a team using the *TWProxy* and the default bots from UT2004. The third experiment shows a battle between a team with *TWProxy* and an human team.

5.2.1 TWProxy-Team vs Machinetta-Team

The aim of this experiment is to assess how better can a team's efficiency be by changing just the proxy that provides the teamwork. Both teams had the same agents and they used the same strategy, the only difference was the teamwork proxy. Figure 6 shows the

Table 2: An example of TTIs measured in one match played by *TWProxy* and *Machinetta*.

Interaction	Elapsed time(ms)	
	<i>TWProxy</i>	<i>Machinetta</i>
-		
01	4	5838
02	5	13307
03	3	3360
04	3	17463
05	4	13256
06	3	12369
07	3	10497
08	4	13358
09	3	14211
10	4	13212
11	4	18211
12	5	17195
13	4	15251
14	4	15138
15	3	9337
16	3	15262
17	4	16251
18	3	15299
19	4	22208
20	4	18207
21	3	23274
22	5	8248
23	4	11264

result of a match, in which the team using *TWProxy* won but the game was very disputed with just two points of difference. The individual agents using *Machinetta* had higher score than those using *TWProxy*, however, the effort of the team should precedence over individual actions. The key point is that the best for the team is not necessarily the best for each individual.

5.2.2 TWProxy-Team vs UT2004-Team

An important point in the use of multi-agent teamwork in modern games is to decide if it is possible to use distributed artificial intelligence in a domain in which nowadays the centralized solution is simpler. The centralized solution can access the full state of game and decide the best to do. However, in a distributed approach, the full state is not accessible, and each entity needs to cooperate in an effort to coordinate their actions. Thus, this experiment confronts these two approaches, the distributed versus the centralized.

The battle was composed by bots from UT2004 in their default skill level and agents using the *TWProxy*. Figure 7 is a example of result in this experiment. The difference in scores was very high in favor of the team with the *TWProxy*, both individually and in team. Therefore, it shows the possibility of usage of the distributed approach which also creates a new possibility of gameplay.

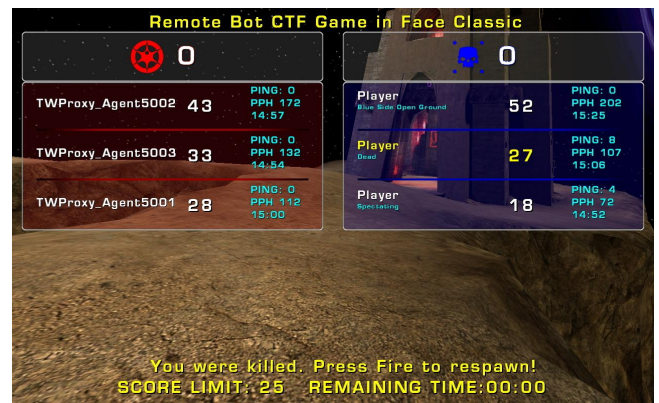
5.2.3 TWProxy-Team vs Human-Team

The last experiment aims to evaluate the team coherence, putting the team with *TWProxy* against a human-team. A human-team has the ability to explore some coherence fault in a team of bots. For this reason, this experiment was interesting.

Although the agents were using a static set of strategies, the human-team failed to explore any teamwork inconsistency. The battle was very disputed even though the human players had average skill in First Person Shooter. Figure 8 shows the result of a match between *TWProxy*-Team and Human-Team, where the balance can be seen.

6 Conclusion

This work shows that it is possible to overcome the deficiencies about social behavior in team-based games using the teamwork approach. With this purpose, this work presents *TWProxy*, a team

**Figure 6:** Results of a Capture The Flag match between *TWProxy*-Team and *Machinetta*-Team.**Figure 7:** Results of a Capture The Flag battle between *TWProxy*-Team and *UT2004Bots*-Team.**Figure 8:** Results of a Capture The Flag match between *TWProxy*-Team and Human-Team.

proxy for dynamic environment with real-time requirements. In order to evaluate *TWProxy*, teams using it were developed, and the agents acted coherently as a team, without to use a central control. Such application creates new possibilities in the gameplay, specially in distributed games like massive multiplayer online.

Using *TWProxy* the group activities are described by a high-level team-oriented programming language, giving more flexibility and autonomy to the team to perform a task. Besides that, agents have a distributed blackboard provided by *TWProxy*, due to the maintenance of beliefs between proxies. The belief description language is also extensible, being able to use in several domains.

The experiments show that it is feasible to use a distributed approach and also the advantages of *TWProxy* face to *Machinetta*. Indeed, the experiments have shown the limitations of *Machinetta* for this specific domain. *TWProxy* had good results in these experiments, showing its stability and efficiency. Besides, *TWProxy* has new important features to team composition, like reuse of plans, easy and extensible language to describe team plan, atomic communication, invariant plans to the number of agents, and extensibility in role allocation using knowledge from a specific domain. *TWProxy* is also more lightweight than *Machinetta*. In the matches experiments, it is possible to conclude that: less time of team inconsistency gives better performance to a group; a distributed approach is feasible and its performance is comparable to team centralized control; and it is feasible to keep the team coherence making believable teams.

References

- ANDREW SCHOLER, G. K., 2000. Gamebots. Último acesso em 01 de dezembro de 2006.
- CHALUPSKY, H., GIL, Y., KNOBLOCK, C., LERMAN, K., OH, J., PYNADATH, D., RUSS, T., AND TAMBE, M., 2001. Electric elves: Applying agent technology to support human organizations.
- DE BYL, P. B. 2004. *Programming Believable Characters for Computer Games*. Charles Development Series.
- DÉFAGO, X., SCHIPER, A., AND URBÁN, P. 2004. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Comput. Surv.* 36, 4, 372–421.
- GROSZ, B. J., AND KRAUS, S. 1996. Collaborative plans for complex group action. *Artificial Intelligence* 86, 2, 269–357.
- GROSZ, B., AND KRAUS, S., 1998. The evolution of sharedplans.
- HILL, R., CHEN, J., GRATCH, J., ROSENBLOOM, P., AND TAMBE, M. 1997. Intelligent agents for the synthetic battlefield: A company of rotary wing aircraft. In *Innovative Applications of Artificial Intelligence (IAAI-97)*.
- JENNINGS, N. R. 1995. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence* 75, 2, 195–240.
- JONES, R. M., LAIRD, J. E., NIELSEN, P. E., COULTER, K. J., KENNY, P. G., AND KOSS, F. V. 1999. Automated intelligent pilots for combat flight simulation. *AI Magazine* 20, 1, 27–41.
- KAMINKA, G. A., VELOSO, M. M., SCHAFFER, S., SOLLITTO, C., ADOBBATI, R., MARSHALL, A. N., SCHOLER, A., AND TEJADA, S. 2002. Gamebots: a flexible test bed for multiagent team research. *Commun. ACM* 45, 1, 43–45.
- KAMINKA, G. A. 1999. The robocup-98 teamwork evaluation session: A preliminary report. In *RoboCup*, 345–356.
- LAIRD, J. E., NEWELL, A., AND ROSENBLOOM, P. S. 1987. Soar: an architecture for general intelligence. *Artif. Intell.* 33, 1, 1–64.
- LEVESQUE, H. J., COHEN, P. R., AND NUNES, J. H. T. 1990. On acting together. In *Proc. of AAAI-90*, 94–99.
- MARSELLA, S., ADIBI, J., AL-ONAIZAN, Y., KAMINKA, G. A., MUSLEA, I., AND TAMBE, M. 1999. On being a teammate: experiences acquired in the design of RoboCop teams. In *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, ACM Press, Seattle, WA, USA, O. Etzioni, J. P. Müller, and J. M. Bradshaw, Eds., 221–227.
- MARSELLA, S., TAMBE, M., ADIBI, J., AL-ONAIZAN, Y., KAMINKA, G. A., AND MUSLEA, I. 2001. Experiences acquired in the design of robocup teams: A comparison of two fielded teams. *Autonomous Agents and Multi-Agent Systems* 4, 1/2, 115–129.
- MONTEIRO, I. M., AND DOS SANTOS, D. A. 2007. Um framework para o desenvolvimento de agentes cognitivos em jogos de primeira pessoa. In *Anais do VI Workshop Brasileiro de Jogos e Entretenimento Digital*.
- MONTEIRO, I. M. 2005. Uma arquitetura modular para o desenvolvimento de agentes cognitivos em jogos de primeira e terceira pessoa. In *Anais do IV Workshop Brasileiro de Jogos e Entretenimento Digital*, 219–229.
- NAIR, R., ITO, T., TAMBE, M., AND MARSELLA, S., 2000. Robocup-rescue: A proposal and preliminary experiences.
- PYNADATH, D. V., AND TAMBE, M. 2003. An automated teamwork infrastructure for heterogeneous software agents and humans. *Autonomous Agents and Multi-Agent Systems* 7, 1-2, 71–100.
- RUSSEL, S., AND NORVING, P. 2004. *Inteligência Artificial - Tradução da segunda edição*. Editora Campus.
- SCERRI, P., PYNADATH, D., AND TAMBE, M. 2001. Adjustable autonomy in real-world multi-agent environments. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, ACM, New York, NY, USA, 300–307.
- SCERRI, P., PYNADATH, D., JOHNSON, L., SCHURR, R., SI, M., AND TAMBE, M., 2003. A prototype infrastructure for distributed robot-agent-person teams.
- SCERRI, P., PYNADATH, D., AND TAMBE, M., 2003. Towards adjustable autonomy for the real world.
- SCERRI, P., JOHNSON, L., PYNADATH, D. V., ROSENBLOOM, P., SCHURR, N., SI, M., AND TAMBE, M., 2003. Getting robots, agents and people to cooperate: An initial report.
- SCERRI, P., PYNADATH, D., SCHURR, N., FARINELLI, A., GANDHE, S., AND TAMBE, M. 2004. Team oriented programming and proxy agents: The next generation. In *Proceedings of 1st international workshop on Programming Multiagent Systems*.
- SCHURR, N., MARECKI, J., TAMBE, M., AND SCERRI, P., 2005. Towards flexible coordination of human-agent teams.
- STEVEN, N. S., 2005. Evolution of a teamwork model. Disponível em citeseer.ist.psu.edu/679191.html. Acesso em: Outubro 2007.
- STONE, P., AND VELOSO, M. M. 1998. Task decomposition and dynamic role assignment for real-time strategic teamwork. In *Agent Theories, Architectures, and Languages*, 293–308.
- SYCARA, K., AND SUKTHANKAR, G. 2006. Literature review of teamwork models. Tech. Rep. CMU-RI-TR-06-50, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, November.
- TAMBE, M., AND ZHANG, W., 1998. Towards flexible teamwork in persistent teams.
- TAMBE, M., AND ZHANG, W. 2000. Towards flexible teamwork in persistent teams: Extended report. *Autonomous Agents and Multi-Agent Systems* 3, 2, 159–183.
- TAMBE, M. 1997. Towards flexible teamwork. *Journal of Artificial Intelligence Research* 7, 83–124.

YEN, J., YIN, J., IOERGER, T. R., MILLER, M. S., XU, D., AND
VOLZ, R. A. 2001. CAST: Collaborative agents for simulating
teamwork. In *IJCAI*, 1135–1144.