Tactical assessment in a squad of intelligent bots

Marcel Gołuński, Piotr Wąsiewicz

Institute of Electronic Systems, Warsaw University of Technology

marcel.golunski@gmail.com

ABSTRACT

In this paper we explore the problem of communication and coordination in a team of intelligent game bots (aka embodied agents). It presents a tactical decision making system controlling the behavior of an autonomous bot followed by the concept of a team tactical decision making system controlling the team of intelligent bots. The algorithms to be introduced have been implemented in the Java language by means of Pogamut 2 framework, interfacing the bot logic with Unreal Tournament 2004 virtual environment.

Keywords: game bots, virtual environment, tactical decision making, influence vectors, pogamut, Unreal Tournament 2004

1. INTRODUCTION

In the age of realistic gameplay and massive multiplayer competition, a squad of intelligent bots requires real-time assessment of tactical situation. Intelligent bot, or an embodied agent, is an artificial player which exists in a Virtual Environment (VE) like Unreal Tournament 2004 (UT). Its role is to substitute a human player by employing artificial intelligence (AI) methods. Our goal is to provide a plausible tactical decision making system, effective for a single bot, as well as a squad of bots. Making contemporary bots competitive and their behavior patterns realistic leads to more entertaining gameplay. It is also a great test-bed implementation for future real-time mobile robots.

Cooperation and communication solutions introduced in the multi-agent systems [1, 2, 3] may be applied to coordinate bots within a squad. The *Agent Communication Language (ACL)* utilized in our work can be a good example of such a solution. Reliable communication channel gives bots the opportunity to achieve common tactical goals. Well known strategic assessment technique called *Influence Mapping* [4, 5, 6] is not suitable for small-scale time-sensitive tactical assessment. *Dynamic procedural combat tactics* [7], which addresses the small-scale tactical appraisal, operates on a discrete, waypoint driven, representation of the game world. The popular *scripting* techniques [8] require a great coding overhead and don't provide desired behavior flexibility. We believe that *Influence Vectors (IV)* technique may fill the gap in that area of bot tactical assessment.

2. TACTICAL DECISION MAKING

The decision making system that we have implemented is divided into two main parts. The *Tactical Decision Making System (TDMS)* of an individual bot is implemented with the use of Finite State Machine (FSM), which is a widely used concept in the game AI industry [9,10]. The *Team Tactical Decision Making System (TTDMS)* is implemented as a separate state in FSM and employs more sophisticated team concepts like communication, maneuvers, formations etc.

2.1 TDMS

The tactical decision making system utilized by each bot consists of a deterministic finite state machine defined as a sextuple $<X, A, Y, \delta, \omega, x_0>$, comprising the following: a finite set of states X, a finite set of input parameters (the input alphabet) A, a finite set of output actions (the output alphabet) Y, a transition function $\delta: X \times A \rightarrow X$, an output function $\omega: X \times A \rightarrow Y$, an initial state $\boldsymbol{x_0} \in X$.

The finite set of states (X) is a predefined list of states representing the internal bot status. States implemented during our experiments are shown in Fig. 1A. They are divided into three categories: *Autonomous states* represent internal status of

Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2010,
edited by Ryszard S. Romaniuk, Krzysztof S. Kulpa, Proc. of SPIE Vol. 7745, 77450W
 · doi: 10.1117/12.872109

an autonomous bot. State transitions are made autonomously based on changes in the virtual environment. *Team states* describe the role of a bot in a team. There are two team states: a leader state, called *Team Controller (TC),* and a team member state, called *Team Bot (TB)*. Team states are superior to autonomous states. In the current implementation of communication channel, there is only one TC available, but this can be easily extended. *Special states* serve special actions, and are managed by the programmer. They were used for the adaptation of several bot parameters, like e.g. the weapon strength factor and for debugging. *The initial state (x_0)* is the entry point for the bot logic. The *Patrol* state was set as initial value.

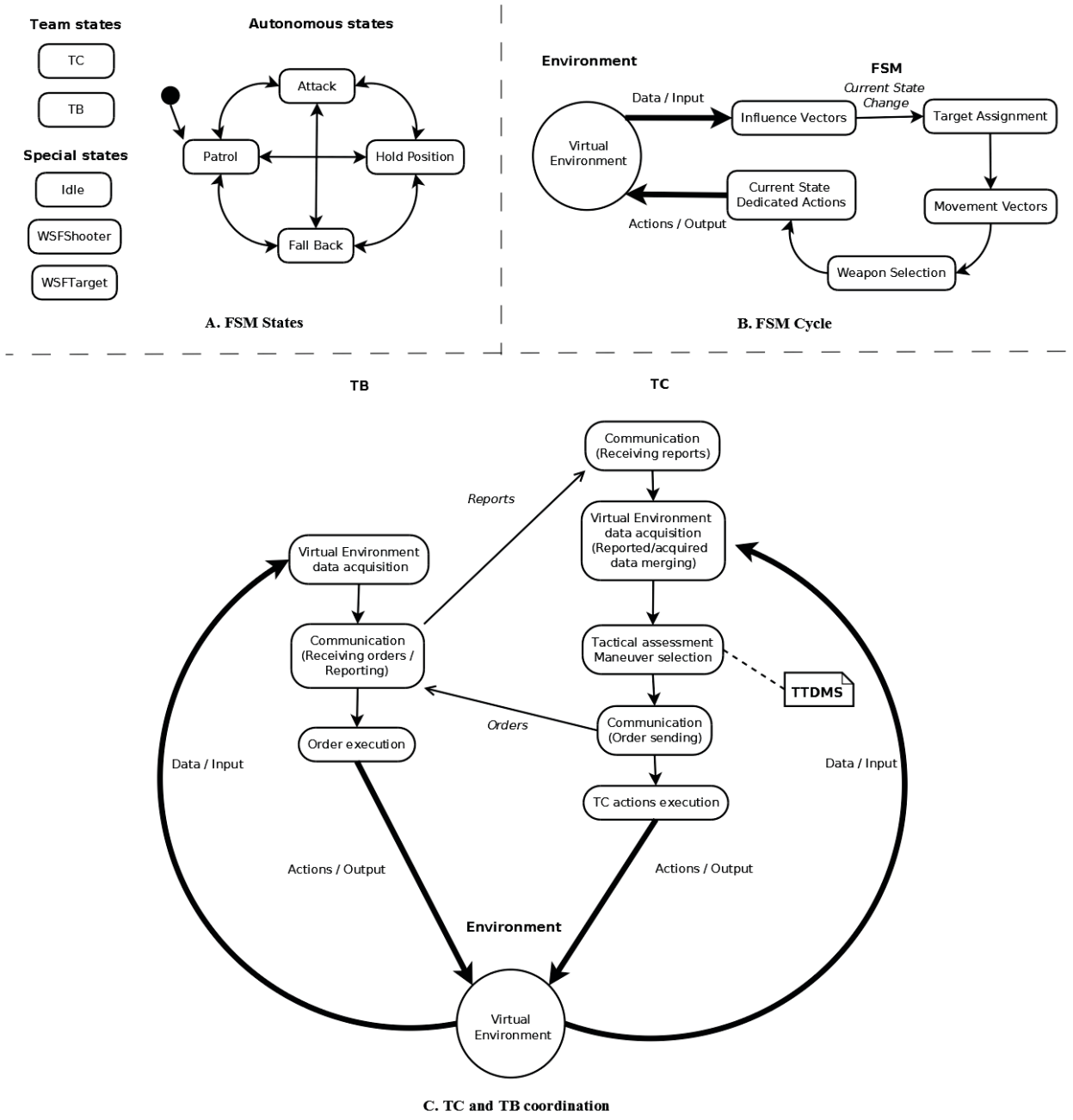

Fig. 1 A: FSM States, B: FSM Cycle, C: Activity diagram for TB and TC states

The finite set of input parameters (A) comprises all the data acquired by the bot from the virtual environment. The said data falls into two categories. *Internal parameters* describe bot status. Their values are available to bot at all times. They include health and armor points, bot location and inventory. Inventory comprises weapons of various types. *External parameters* are variably and occasionally available to bot. They are composed of world objects data (e.g. location of other players), team messages delivered by a communication channel and programmer commands sent via the UT messaging system. Both internal and external parameters add up to form the "input alphabet", which determines bot state transitions.

The finite set of output actions (Y) contains all actions available to bot, which include movement and two types of attacks. Attack is performed with the use of one weapon selected from the bot's inventory. Bot can move to a designated location specified by a three-dimensional Cartesian coordinate or head for a specific named player. Additional bot movement actions include jumping and dodging. The bot model is subject to VE gravity and other interactions.

The transition function (δ) determines internal bot state changes based on the input parameters. In the autonomous variant, where bot logic is in one of the autonomous states, transitions are made based on the *Influence Vectors (IV)* method, which is discusses at a later stage herein. In the team variant, there are no state changes made by bots. The bot stays in one of the team states until directed to switch to another state. The team state transitions work as follows: if the communication channel is available, bot designated to be a leader switches to the *Team Controller (TC)* state. Team members register in the TC squad list through the communication channel. If registration is successful, the connected bot switches to the *Team Bot (TB)* state and awaits orders from TC.

The output function (ω) decides which actions to perform. In the case of individual bot, output function is divided into four separate steps: *target assignment*, *movement*, *weapon selection* and the *current state dedicated actions*. The target assignment algorithm picks the best out of available targets based on the influence value given by the IV method. Movement algorithms attempt to provide a tactical advantage, e.g. by minimizing the points of contact [6] with the enemy. Weapon selection is made with respect to the weapon characteristics represented by the *Weapon Strength Factor (WSF).* The current state dedicated actions enclose all other state-dependent actions connected with the current state purpose.

As described above, the FSM works in cycles (Fig. 1B). Each cycle starts with the data acquisition from the VE. These data constitute the bot logic input. The input data are processed by transition and output functions, which leads to actions performed by the bot in the VE. Those actions constitute the bot logic output. The FSM is a robust and flexible abstraction allowing for a straightforward addition of numerous states, e.g. connected with other game types. Two of the FSM states are strategically significant: they make up the team logic and, therefore, are called the team states.

2.2 TTDMS

The team tactical decision making system is based on the TDMS as described in the previous section. The bot team logic is implemented using dedicated team states: TC and TB. The current state dedicated actions are executed at the end of the FSM cycle; therefore, TC and TB state actions can override the individual bot logic. To ensure a proper coordination in a squad of bots, the following concepts should be taken into account. *Communication* allows for exchange of various VE data between the bots. In our team, tactical decision making system communication is based on the FIPA-ACL language [3]. We have provided our own ACL implementation based on the FIPA-ACL specification. The language makes use of a simple content language and simple ontology. The use of said concepts makes communication language flexible and extensible. *Organizational structuring* plays a key role in the squad coordination. We employ centralized coordination model with mixed command style [11, 12] with a single team leader called Team Controller (TC). If a more complex team structure were required, command hierarchy [13] could be easily introduced. *Maneuvers and formations* were used to control the team movement. Maneuvers [12] were implemented as predefined sets of actions that help coordinate the squad movement in a strict and planned manner. Maneuvers can incorporate formations [14] in order to provide suitable line-up of bots. *Synchronization* is a very important factor in bot coordination. Given the test platform limitations, we have implemented loose coordination of bots; therefore, issues concerning synchronization fall out of the scope of this paper.

Fig. 1C illustrates coordination cycle between TC and TBs. The bot squad consisting of one leader (TC) and a number of team members (TBs) executes coordinated actions. Each TB acquires data from the VE. The data comprise enemy locations and TB status. Such data are sent to TC as a report. TC receives reports from TBs and merges the received data

with the data acquired from VE by himself. The merged data are used for tactical assessment and proper maneuver selection, handled by TTDMS. After the proper maneuver is selected, TC sends orders for TBs and executes its own actions. TBs receive orders from TC and execute them.

The main task of the team tactical decision making system is to select the best maneuver for the current environment state. The maneuver is selected from a finite maneuver set. The maneuver selection is performed according to the following formula:

$$h : S_E \rightarrow M_{db} \tag{1}$$

where: h is the TTDMS hypothesis, S_E is a set of parameters describing all possible environment states, M_{db} is a maneuver database (set).

Fig. 2 presents a simplified TTDMS decision cycle. The most appropriate maneuver $m_i \in M_{db}$ is selected, according to hypothesis h, based on the current environment state $s_E \in S_E$. The environment state is a set of continuous parameters describing enemy forces location, team location and other parameters, which, to be presented in a more computation suitable manner, were subject to the *Influence Vectors* method aimed at limiting the number of parameters. This limited number of continuous parameters was then discretized to a limited symbolic set representing VE states. The formula (1) can be presented as:

$$h : S_{EV} \rightarrow M_{db} \tag{2}$$

where: h is the TTDMS hypothesis, S_{EV} is a finite, countable set of discretized parameters, representing the environment state, M_{db} is a maneuver database (set). The formula (2) is suitable for applying a learning algorithm to provide the TTDMS hypothesis h.

We have introduced a simple reinforcement learning (RL) algorithm [5] that we have called *Score Statistic Reinforcement Learning (SSRL)*. Its goal is to maximize immediate rewards; therefore the RL discounting factors have been zeroed. The reinforcement value is calculated by subtracting the enemy score from the team score achieved over a given time span. The subtraction result is called the score ratio. Average, minimum and maximum score ratios are stored in the algorithm's data structures along with the number of repetitions for each of the actions. The algorithm uses the state-action value function. This function is presented as formula (3). This value function has been dubbed the *score evaluation*.

$$S_{eval}(s,m) = \left(S_{avg}(s,m) + \alpha\left(S_{\max}(s,m) + S_{min}(s,m)\right)\right) \cdot \left(\frac{n(s,m)}{N(s)}\right)^{\beta} \tag{3}$$

where: s is the current VE state, m is the selected maneuver, S_{eval} is the value function, S_{avg} is the arithmetic mean of received reinforcement, S_{max} is a maximal reinforcement value, S_{min} is a minimal reinforcement value, N is the largest number of conducted tests for a given VE state, n is a number of tests for a given maneuver in a given VE state, α determines the influence of minimal and maximal reinforcement on the value function, β determines the influence of number of conducted tests on the value function.

3. INFLUENCE VECTORS

The key concept underlying the tactical assessment used in our work is the *Influence Vectors* (IV) method. This is our proposal of an algorithm capable of tactical assessment in a dynamic environment. This assessment is limited to an engagement area usually being a convex area. Fig. 3 depicts the essential idea behind the influence vectors method. The IV method is a modification of a well known technique called influence mapping [4,5]. In the influence mapping method, a set of square cells is superimposed on the game world and the influence is calculated for each of those cells. The influence spreads from cell to cell, starting from the cells occupied by units, propagating centrifugally. Influence value, passed to adjacent cells, diminishes with each progression by a given factor. In the IV method, the influence is calculated not for the game world cells but rather for the units themselves. Each unit in the VE influences one another with an influence value I as depicted in the Fig. 3. Some influence might be positive (of a friend) other might be negative (of an enemy). The *Total Influence* (*TI*) factor assesses bot tactical situation in the engagement area and is given by the equation:

$$TI = C_F - \sum_{e=0}^{E} I_e + \sum_{f=0}^{F} I_f \quad (4)$$

where: *TI* is a total influence factor, C_F is a condition factor (e.g.: health + armor), *E* is the number of enemies, *F* is the number of friends, *I* is the influence of: *e* enemy, *f* friend. In the simplest approach if *TI* factor is below 0 the bot is in a very bad situation, if *TI* is high it has a tactical advantage. Influence *I* can be expressed with the following simple equation:

$$I = S \cdot R \quad (5)$$

where: *I* is the influence value, *S* is a unit strength factor, *R* is a relation factor between the bot and a target player which influence is taken into account. The strength factor can be of any type, but it must describe the combat power of a target unit. In our experiments, we have introduced the *weapon strength factor,* which describes the strength of a unit. This strength is described by the firepower of a weapon a unit wields, considering weapon damage, distances between the adversaries, etc. The relation factor can modify the strength of a target unit by any factor that can originate from a tactical advantage, e.g. terrain advantage, or an interaction level between the bot and the target unit. The technique in question can be used to determine many tactical aspects of a combat like action selection, target assignment, team support. Action selection can be inferred on the basis of TI value. Low TI values suggest defensive tactics, whereas high TI values impose offensiveness. Altering TI threshold values can model the behavior of a bot. Different threshold level sets can shape diverse behavioral models, ranging from placid to aggressive. Target assignment based on the influence values can be of several types: including targeting the weakest or the strongest enemy based on their TI value; targeting the greatest threat based on the largest influence constituent of bot TI factor. TI factor of a friendly unit can be used to ascertain assistance requirements. Too high TI values of nearby friendly units can hint the bot logic to reduce the proximity with other team members. Too tight formations are vulnerable to area damage attacks and are also accidental fire prone.

4. IMPLEMENTATION

Implementation of the aforementioned algorithms was done using the Pogamut 2 platform [15], which is the interface between the bot logic, coded in the Java language, and the virtual environment of Unreal Tournament 2004. Fig. 4 depicts the relationship between the game world and the bot logic. Pogamut platform is a rapidly developed project, which allows AI programmers to create *Intelligent Virtual Agents* (IVA), referred to as bots in this paper. Agent behavior can be observed in a real-time three dimensional environment. Pogamut platform provides several modules of bot logic, most significant of which being the *AgentBody,* which allows the programmer to control the bot behavior, and the *AgentMemory,* which provides information on the bot state, as well as information on what the bot has seen.

We have provided a communication module which incorporates PComm module supplied by the Pogamut 2 developer team. We have implemented the ACL library and extended the bot logic with the required features comprising the aforementioned methods. The core of the bot logic is the TDMS and TTDMS shown in Fig. 4. The entire programming was done using the Java language. The Java code is executed within the NetBeans IDE JVM, which is provided with the appropriate plugin allowing the programmer to add and control the bots connected to the game server. The platform supports fully distributed access to the game server owing to the GameBots2004 UT extension which allows remote access to the game world.

5. EXPERIMENTS

We have conducted a number of experiments to test the bot logic performance. The experiments covered the encounter between UT2004 *Native Bots* (NB) and our autonomous bot called *Influence Vectors Combat Bot* (IVCB). The IVCB employs the TDMS. We have also tested the efficiency of our team bot called *Team Combat Bot* (TCB) against NB and IVCB. TCB utilizes the TDMS and TTDMS. Experiments were run in a simple test-bed setting. Bots contended in an almost square, large convex engagement area. Given the influence vectors method intended application, there was no need for a more complex scenario. In the said experiments, two teams of bots competed in order to reach the team score of 100 points. Individual bot scores are summarized as a team score. All bots are characterized by the skill level, which, in the case of our bots, influences the shooting ability. In case of native UT bots, the skill level also determines bots agility and tactics level. The experiments were conducted with a view to covering various skill levels.

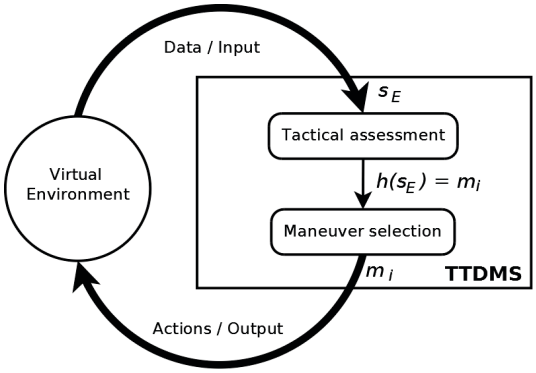

Fig. 2 TTDMS cycle

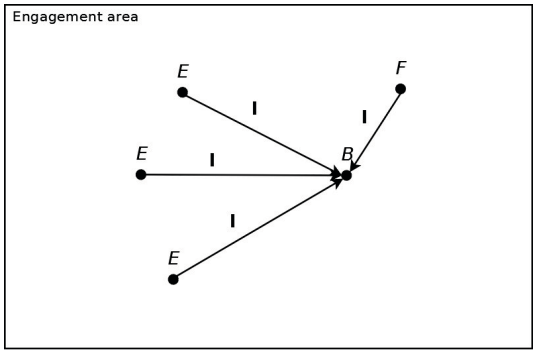

Fig. 3 Exemplary influence vectors

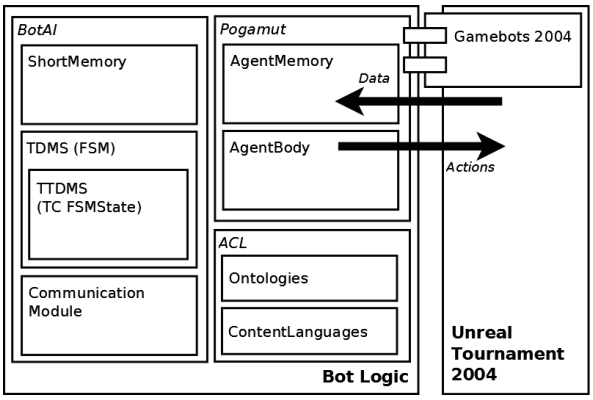

Fig. 4 Bot logic

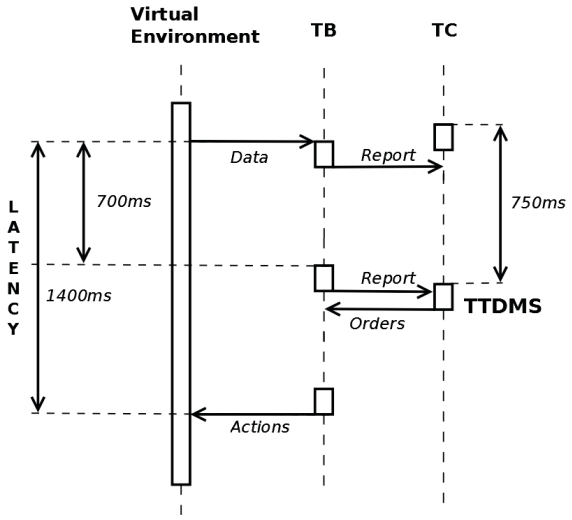

Fig. 5 Latency of order execution

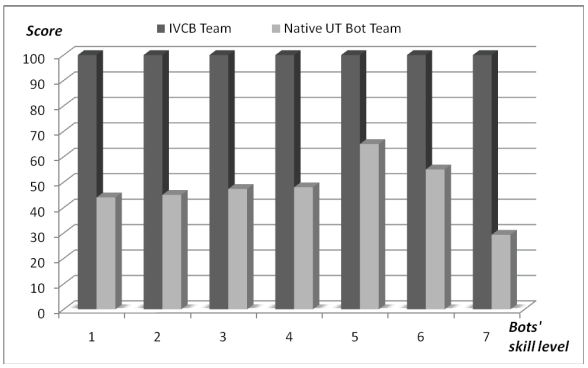

Fig. 6 The score result of the confrontation between IVCB and NB in function of bots' skill level

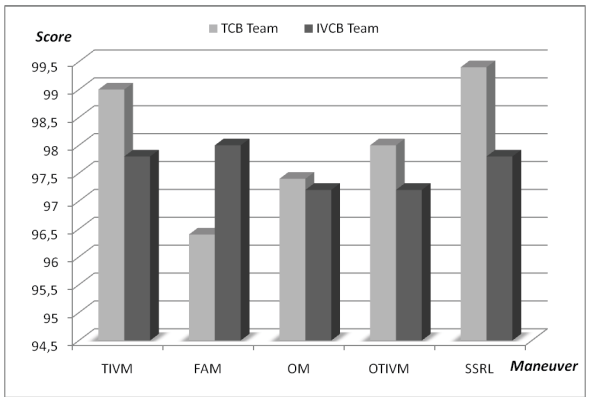

Fig. 7 The score results of the confrontation between TCB and IVCB in function of the maneuver selected

In our experiments, we have encountered several problems, the key issue at stake being the *latency of order execution* problem. Fig. 5 depicts the worst case scenario illustrating the above. In the squad of bots, where the AI logic is distributed among many agents the report and order delivery times are vital. If the bot logic cycle, comprising logic

execution and logic interval, takes too long, the reports on environment state, as well as orders, may be outdated. We have come up with the following conclusion: *Bot cycle step strongly depends on the VE significant step in a case of centrally coordinated decisions*. To solve the latency of order execution problem, we have changed the command style from authoritarian to mixed insofar as to allow every TB to assign the target by itself when the target coming from TC's order was invalid. The most important measure, however, was separation of testing platform elements. We have put the VE itself on the different machine than the bot logic, which substantially shortened the bot logic interval, thus resulting in a much better score.

The exemplary results of the confrontation between IVCB and NB are shown in Fig. 6. The exhibit presents the average score result for 5 repetitions in a 5-player team deathmatch. IVCB proved much better teamwork capabilities, although the bot logic does not contain explicit squad routines. The team behavior is an implicit outcome of an influence vectors method encouraging a group of bots to attack isolated enemies. The bots TI factors rise when more friendly bots are attacking single, secluded enemy. This kind of cooperative behavior forms an emergent maneuver [11]. IVCB achieved victory despite the bots skill level, which proves that the influence vectors technique is efficacious. Fig. 7 shows the results of a skirmish between IVCB and TCB in a 4 player team deathmatch. The score is the average value for 5 repetitions. Each column pair represents score gained by TCB and IVCB for a different maneuver used by TCB (IVCB employs only the TDMS and do not contain any explicit team routines). The last column pair shows the score result of the TTDMS learned with the SSRL algorithm (picking the best maneuver for the current VE state). The result shows that, in most cases, the combat capabilities of TCB are better than those of IVCB. The explicit team factor improves the overall performance of a squad. The score result would probably be much better for the TCB team if the test scenario were more complex, and bots had to coordinate their behavior in a vast network of rooms.

6. CONCLUSIONS

The goal of this paper was to present an exemplary realization of tactical assessment in a squad of artificially intelligent bots. Teamwork proved to be an important factor even for a simple test-bed scenario. Realistic gameplay and plausible team behavior prove to be pivotal features in the contemporary gaming experience. We believe that the techniques presented may also be used with future mobile robots which may need to coordinate their actions in order to achieve common team goals.

The possible future work includes: fully distributed environment (each bot on a separate machine), maneuver recognition and discovery, methods for synchronizing bots behavior in a distributed environment, testing the methods in more complex scenarios, extensions to the influence vectors method and further research on other tactical assessment algorithms.

REFERENCES

1. Aart van C., *Organizational Principles for Multi-Agent Architectures,* 139-176, Birkhäuser Verlag, Basel, 2005.
2. Weiss G., *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence,* 79-96, The MIT Press, Cambridge, 1999.
3. Bellifemine F., Caire G., Greenwood D., *Developing Multi-Agent Systems with JADE*, 13-27, John Wiley & Sons Ltd, 2007.
4. Tozour, P. „Influence Mapping", *Game Programming Gems 2*, DeLoura M., II, 287-297, Charles River Media, 2001.
5. Millington I., *Artificial intelligence for games,* 499-532*,* 612-628, Morgan Kaufmann Publishers, San Francisco, 2006.
6. Woodcock S., „Recognizing Strategic Dispositions: Engaging the Enemy", *AI Game Programming Wisdom*, Rabin S., I, 221-232, Charles River Media, 2002.
7. Straatman R., Sterren van der W., Beij A., "Killzone's AI : Dynamic Procedural Combat Tactics", *Game Dev. Conf.*, San Francisco, 2005.
8. Rabin S., *AI Game Programming Wisdom*, 503-554 (chapter „Scripting"), Charles River Media, 2002.
9. Dybsand E., "A Finite-State Machine Class", *Game Programming Gems*, DeLoura M., I, 237-248, Charles River Media, 2000.
10. Carlisle P., "Designing a GUI Tool to Aid in the Development of Finite State Machines", *AI Game Programming Wisdom,* Rabin S., I, 71-77, Charles River Media, 2002.
11. Sterren van der W., "Squad Tactics: Team AI and Emergent Maneuvers", *AI Game Programming Wisdom*, Rabin S., I, 233-246, Charles River Media, 2002.
12. Sterren van der W., "Squad Tactics: Planned Maneuvers", *AI Game Program. Wisdom*, Rabin S., I, 247-259, Charles River Media, 2002.
13. Reynolds J., "Tactical Team AI Using a Command Hierarchy", *AI Game Programming Wisdom*, Rabin S., I, 260-271, Charles River Media, 2002.
14. Dawson C., "Formations", "*AI Game Programming Wisdom*, Rabin S., I, 272-281, Charles River Media, 2002.
15. Kadlec, R., Gemrot, J., Burkert, O., Bída, M., Havlíček, Brom, C., "POGAMUT 2 - A platform for fast development of virtual agents' behaviour", *Proceedings of CGAMES 07*, La Rochelle, 2007.