vBattle: A new Framework to Simulate Medium-Scale Battles in Individual-per-Individual Basis

Luis Peña, Sascha Ossowski and José-María Peña

***Abstract*—Strategy games such as Warcraft™ or UFO™ franchises or RPG games like Never Winter Nights™ or Baldur Gate™ are successful blockbusters in video game industry. These games are based on battles simulated individual per individual. These type of games is a very interesting scenario to develop multilevel strategies or emergent behavior in multiagent systems.**

This paper presents a new computational intelligence framework, named VBATTLE, for the evaluation of learning strategies in video games. The framework simulates a battle game in which two or more contenders are fighting in units with a high-detail individual-per-individual resolution. This simulation considers aspects of (1) actions parameters (action time, exhaustion consumption), (2) non-deterministic action resolution, (3) hierarchical intelligence (individual vs. unit strategies), and (4) scenario interaction.

The VBATTLE framework is designed to have both a 3D visual representation of executions (either on-line or post-mortem visualization) as well as a server-based engine to perform learning tasks. This contribution presents the design principles of VBATTLE framework, its objectives, the possible applications for developing computational intelligence algorithms. In addition, preliminary results using a limited version of the framework are also presented.

I. INTRODUCTION

The game industry has been working, since the creation of games such as Romance of the Three Kingdoms series™ (early 80s) or Wizard's Crown™, in the adaptation of different levels of strategy and tactical confrontations. These, and other, games clearly show the evolution of artificial intelligence mechanisms to solve combat simulations. The principles in combat simulation are based on the statement that assumes that every actor (playing characters, enemies, etc.) has a set of actions that triggers according to the current state of the game or scenario, in the most simple schema the actors have only the attack and the defense actions.

Usually, there are two levels of combat simulation: (1) the strategic combat which computes the result of a combat summarizing the strengths of the sides involved in the combat and applying some factors such as the terrains or so, for example Civilization Series™ and Age of Empires™, and (2) the tactical combat represented as a deployment of troops across a battlefield which take more detailed decisions about their movement in combat, and use some specialized action such as fire projectile weapons, for example in the Total War Series™ or even the UFO™ franchise. Between these two levels, there are only few games (e.g., Dominions Series™) applying the possibility of simulate the tactical combat in order to solve large-scale strategic combat. This circumstance avoids the use of more detailed actions to a group of actors (a unit) based on the individual characteristics that make their strategy more appropriate to face a given enemy. From the perspective of computer-controled characters, this would represent a more adaptative behavior that makes game experience more atractive for users.

The strict deadlines of the game industry many times leads to a sort time for the investigation in new techniques, mainly in artificial intelligence, and only step by step the new algorithms and methods appear in commercial games. Usually, the game developers are more interested on designing a superb graphical engine and reuse some of the old tailored algorithms for the decision making. To overcome this, in the last few years, the industry provides the game engines with the modding and scripting tools to affiliate permanent gamers and to expand the capabilities of the game; the results, sometimes, are some new implementations of algorithms and techniques which increase the performance of the intelligence engines.

These modding utilities are applied to the research in artificial intelligence [1], [2], in some cases the script programming of the industrial game engine can be a good visual appeal workbench, providing qualitative and, sometimes, quantitative results for the experiment. But, usually, this type of engine: (1) does not provide a complete set of result analysis toolkits, (2) it presents a heavy-weight engine that does not support massive training needed for some algorithms and techniques, being evolutionary algorithms or reinforcement learning clear examples, and (3) the modding tools are limited in terms of customization or flexibility. These restrictions sometimes leads to the development of hand-made engines such as RoboSoccer[1] or MiniGate[2] that are motivated by the specific purpose of research in artificial intelligence, enabling the analysis of results and ad-hoc implementations for different research purposes. Supporting these ideas, and with the intention of providing a new framework for the research of applied artificial intelligence techniques, the VBATTLE Framework is born, a new environment to work in AI methods that need some specific setting such as massive execution, multilayered decision making,

L. Peña and S. Ossowski are with the Centro para las Tecnologías Inteligentes de la Información y sus Aplicaciones (CETINIA), Universidad Rey Juan Carlos, Móstoles, Spain (email:luis.pena,sascha.ossowski@urjc.es)

J.M. Peña is with the Departamento de Arquitectura y Tecnología de Sistemas Informáticos (DATSI), Universidad Politécnica de Madrid, Madrid, Spain(email: jmpena@fi.upm.es)

[1] http://www.robocup.org/

[2] http://ticc.uvt.nl/ pspronck/minigate.html

978-1-4244-4815-9/09/$25.00 ©2009 IEEE

emergent behaviors, etc.

The document is organized as follows: Section II analyzes some of the existing current frameworks, suitables for AI testing, Section III introduces the vBATTLE framework, the objectives and features that it has and the architecture of the solution. In section IV, some of the computational applications on the artificial intelligence experimentation are presented. Section V briefly explains a set of experiments already done using the preliminary implementation of the framework. Finally, section VI concludes with the open issues and the future work.

II. CURRENT FRAMEWORKS

Looking at some of the current frameworks available on the computer games world we obtain some features desirable for the creation of our new framework, at the same time we realized that the current frameworks do not meet all of our specifications for the further researches that we want to carry out.

The features that current frameworks have, and we want to mantain are:

- the bulk of the AI code as a separate library to be engine-independent.
- use console-based controls of the application for independent tests.
- integrate working features into whichever engine you feel most comfortable with.

The idea is that we do not want to be too dependent on any engine, we need a flexible engine that works with different technologies in order to test a set of algorithms and we want to have a control over the game variable to analyze the results and the progress of the different techniques.

A. NERO 2.0

The NERO [3] is a computer game that recreate a tactical combat of adapting intelligent agents, evolving a robot army by tuning their artificial brains for challenging tasks. The learning agents in NERO are simulated robots that are trained as a team of these agents for combat. The agents begin the game with no skills and only the ability to learn. In order to prepare for combat, the player must design a sequence of training exercises and goals. At the training phase AI agents evolve in real time while the game is being played, the player's role is to train the AI for competition.

The core technology of NERO is a special real-time version of NeuroEvolution of Augmenting Topologies (NEAT), providing to the AI agents a real neural networks that are continually growing more complex as the game is played. It allows the AI agents not to be prepackaged or scripted; rather, their behavior emerges in reaction to how the game is played.

1) Pros:

- NERO has a great appeal and can be an excellent vehicle for demystifying AI and demonstrating its value to the public.
- NERO presents the level of the agent decision and the unit decision as an emergent behavior of the agent.
- NERO has two different phases, training and competition.

2) Cons:

- NERO needs that the user interacts with the system in the training phase.
- NERO only uses neural networks for the learning, it does not provide a way to insert another algorithms.
- NERO is not open-source to analyze the code.

B. ORTS

ORTS [4] is a programming environment for studying real-time AI problems such as path finding, dealing with imperfect information, scheduling, and planning in the domain of Real-Time Strategy (RTS) games. Commercial RTS games are non-open-source software which not facilitates researchers to connect remote AI modules to them.

These games are fast-paced and very popular. Furthermore, the current state of RTS game AI is bleak which is mainly caused by the lack of planning and learning - areas in which humans are currently much better than machines. Therefore, RTS games make an ideal test-bed for real-time AI research.

Commercial RTS games are based on peer-to-peer technology - which in a nutshell runs the entire simulation on all player machines and just hides part of the game state from the players.

The ORTS project creates a free software system that lets people and machines play RTS games. The communication protocol is public and all source code and artwork is freely available. Users can connect whatever client software they like. This is made possible by a client/server architecture in which only the currently visible parts of the game state are sent to the players. This openness leads to new and interesting possibilities ranging from on-line tournaments of autonomous AI players to gauge their playing strength to hybrid systems in which human players use sophisticated GUIs which allow them to delegate tasks to AI helper modules of increasing performance.

The ORTS server is responsible for simulating unit actions and determining what each player is allowed to know about the current state of the world. ORTS has a built-in tournament manager, which significantly reduces time for the design of experiments and analysis of results

1) Pros:

- ORTS use a client/server architecture that leads to decouped systems that enable unattended executions or personalized clients.
- ORTS has a simple scripting language for game specifications and GUI customization.
- ORTS players can connect whatever client software they like to and can issue commands to all of their units in each game tick.
- ORTS has a AI-components pluggable system to delegate some AI task via messages to the client which can implement modules to perform these actions.

2) Cons:

- ORTS plays in real-time, which is not suitable for training process of some interesting algorithms that require some offline tuning.

C. Lux Delux

Lux Delux is a game of strategy and domination inspired by the board game Risk. Control armies to conquer and hold strategic countries on the map. Lux Delux provides a SDK with which programmers can create AIs and random map generators. It includes API documentation as well as the source code to all the AIs that are shiped with Lux, and some map generators. The programming language used is Java, but the map generator interface can be used in other languages, as Perl.

1) Pros:

- Lux use a client/server architecture that leads to decoupe systems that enable unattended executions or personalized clients.
- Lux has an unattended play mechanism that can me run in a fast pace, enabling training.
- Lux has a Java SDK API for AI programming.

2) Cons:

- Lux is commercial, and not open-source.
- Lux presents a very simple units without a customization mechanism.
- Lux has an unique display mechanism.

D. Neverwinter Nights™ Series

Neverwinter Nights ™Series is a commercial hit of Bioware, now it is a classic, a reference of Computer Role Playing Games (CRPG), it supports Dungeons & Dragons pen-and-paper rule set. NWN has a high modularity, enabling the public expansion of its campaigns and games.

NWN gives programers all the tools needed to build customized modules, campaigns, and adventures - create buildings, terrain, script encounters, write dialogues, create quests and items - with its Aurora Toolkit .

1) Pros:

- NWN Series can use a client/server architecture that enable unattended executions, controlled by AI.
- NWN Series has a robust, well tested, engine, supported by a large company.
- NWN Series has scripting language, very similar to C++, suitable for AI programming.
- NWN Series is an actual commercial games.

2) Cons:

- NWN Series is commercial, and not open-source.
- NWN Series has no extensibility outside of the Aurora Toolkit, not allowing the development of trace or display plugins.
- NWN Series runs in real-time, not supporting long training algorithms.

III. vBATTLE FRAMEWORK

vBATTLE is a video game framework based on the Java game engine designed as an independent light-weight event-driven simulator with a decoupled visualization tool.

A. Objectives and Features

The vBATTLE framework has been designed following several principles and objectives:

1) **Discrete event simulator**: The vBATTLE framework are based on a discrete event simulator (DES) (as opposite to turn-based or real-time simulations). DES simulates sequences of events based on execution time, in the case of vBATTLE, the different actions require a given amount of time to be performed. Once an action is selected for execution, a new event is created and issued when the action will be finished (when its effects are applied).
2) **Parameters, actions and actors**: Combatants in vBATTLE have different status counters (life, exhaustion, stunning time), and a set of possible actions (that can be different from one actor to another). The actions are also parametrized based on their effects, the execution time, the chance of success, exhaustion points, or other parameters (amount of damage or defence factor).
3) **Non-deterministic action system**: Actions are not deterministic, thus the execution of an attacking action has a probability of success. Defensive actions performed by the target also modify this chance of success or indicate different results and effects.
4) **Scenario interaction**: The scenario is divided into hexagonal cells. The actors are fighting on this interactive scenario that modifies different factors, such as mobility rates, penalties for action executions and exhaustion modifiers. Scenarios may also include restricted movement areas and dangerous or hazardous cells.
5) **Multilevel intelligence**: The vBATTLE framework allows the definition of two level of strategic game, (1) one is individual-centered intelligence, controlling the actions performed by a single combatant, based on the information it receives, and (2) the other is unit-based controllers, in which a number of combatants are managed by a single intelligent mechanism that decides movement of troops and coordination issues, delegating fine-grain action selection to the previous level.
6) **Decoupled Interface**: vBATTLE presents a clear division between the simulation engine and the graphical interface, this feature provides off-line simulation of a large amount of combats, i.e. for learning in some algorithms, and a visualization interface that enable the possibility of view the combat execution or even the further combat analysis.

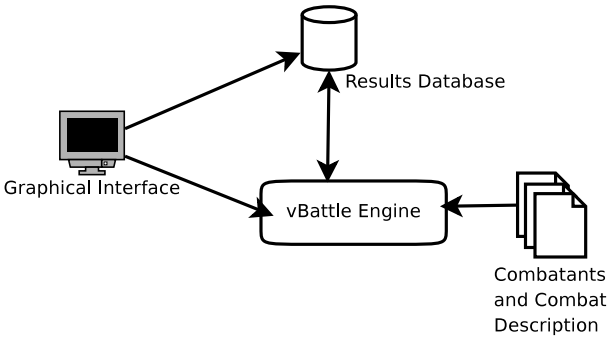

Fig. 1. General Architecture.

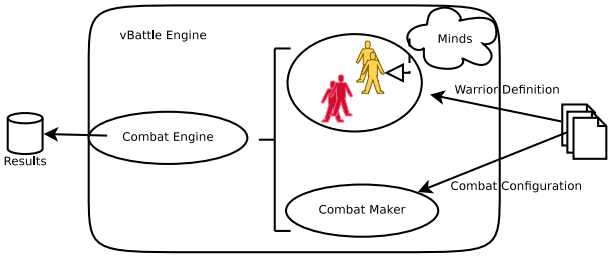

Fig. 2. Engine Decomposition.

B. vBATTLE *Architecture*

The vBATTLE framework is organized as a light-weight event-driven simulator, designed to run in an independent server for a large number of combat simulations (Fig. 1). The engine of the framework receives the configuration loaded from the setup sources, that describes the character's parameters and actions (**Warrior Definition**), combat setup (**Combat Configuration**) and the type of combat and minds (controllers) which we want to use. Once the engine starts, retrieving the relevant information that it needs from the database, such as previous learned statistics, it runs a complete battle, uploading the results to the database (Fig. 2).

The GUI can be executed from a client whenever it is needed, with the option of viewing the current execution running on the engine or loading a previous combat stored in the database. It displays an hexagonal grid with the positions, movements and actions that the combatants do (or did).

Once the GUI is connected, it controls the simulation speed, in order to be able to visualize the combat properly.

The components of vBATTLE in some detail are:

1) **Combat Configuration and Warrior Definition** are presented as a set of configuration files that provide information about the type of combat we want, the character's description and the information about the group of characters. These files could also be extended to support new algorithms. In the current version the configuration files are property files, the future versions could include a XML configuration schema.
2) vBATTLE **Engine** supports the combat simulation, it runs the DES sequence treating all the configured controllers evenly. It applies the rules of the tournament and the stop criteria.
3) **Database** stores: (1) the results of the combats, (2) the evolution of the warrior state and (3) the learning parameters of the agents. The database storage runs in parallel to the engine executing the queries in a batch process. The vBATTLE framework supports different database managers such as MySQL o SQLite.
4) **Graphic Interface** and analysis tools are support modules that provides graphical representation of the combat execution. Preliminary versions of vBATTLE include a simple GUI and basic reporting and statistical tools. Nevertheless, this set of tools are open to the plugging of new modules with more sophisticated analysis (data mining, for example).

C. *Combatants and Action Description*

A key element in the simulation is the combatant's profiles (characteristics, actions and controller). A combatant is each character that has to interact with the environment and other characters performing actions. A combatant is basically a software agent parametrized by some particular information, included in its description.

1) Combatants Description: Every character has two state counters, **Hit Points** (`HPs`) that represent the remaining life for this character and the **Exhaustion Points** (`EPs`) counter that shows the fatigue level of the character. If HPs reach 0 the character is dead and thus it is defeated. On the other hand, if EPs are below 0 the character cannot do anything but rest until it is recovered.

In addition to these characteristics, the combatants have information about their movement, long-range attacks and further modifications to the actions and maneuvers that will affect to the maps and group interactions.

The combatants are described by separate documents which contain the characteristics and actions. This description is independent of the mind that controls the character, allowing the user to create tournaments based on the same warrior description which only differs on the way to choose the actions along the combat (the controllers/minds). This approach is an appropriate test-bed for different computational intelligence algorithms.

A character can perform three types of actions: Offensive, Defensive and Miscellaneous Actions.

- **Offensive Actions** take a fixed amount of time to be executed, named Action Points (`APs`), which represent the time that the action takes to be triggered after it is called. In addition, Offensive Actions consume some EPs when they are triggered. Once an Offensive Action is fired it has a probability of hitting the target and inflicting some damage. The damage of an action can be of three types: (1) HPs damage, (2) EPs damage and (3) Stun damage. The first two damage types represent a direct amount to be substracted to the respective counter of the enemy. The stun damage works in a different way: this type of damage makes the target to cancel his

present declared action and makes that the target cannot declare any other action until he gets recovered from the stun.

- APs of **Defensive Actions** represent the time the defense is active when it is declared, consuming the EPs when it finishes. If a character is hit and he has a Defensive Action declared, he has a probability of blocking the attack. If the Defensive Action blocks the attack, the damage taken by the character is reduced by a factor applied to the HPs and EPs damages and, if the Defensive Action specifies it, to the stun damage.
- **Miscellaneous Actions** are a group of possible actions that provide modifiers or perform scenario-specific effects. An example of a miscellaneous action is to draw a weapon or to load a range weapon. As in the example of offensive actions, the APs indicate the amount of time required to perform the action.

When an action is triggered, it is solved depending on its type, and then the character choices another action if he has any remaining EPs. If he has none, the character must rest for a fixed amount of time to recover some EPs.

D. Engine Description

The **Combat Engine** runs the combats with a continuous round mechanism, every time an event occurs the engine evaluates it, resolving the attacks, deciding a new action, recovering EPs, etc.

As it is said, there are three main action types, attacks, defenses, and miscellaneous. The first difference (skipping the obvious) is that the attack and miscellaneous actions are declared and take a fixed APs to occur and when the activation time reaches the action is executed; and the defense actions last from the declaration instant and during the APs they have.

The attack simulation considers the attacker and the target and the action performed, evaluating if the attack hits the target and computing the damage done, as well as any side effects (such as stunning), it is reduced if the target had a defense declared and it works.

When a combatant has no action selected to be executed, because he has just finished to do something or he has recovered from stunning damage, the mind that controls the character must provide a new action to the engine. The decision from this controller is scheduled based on a continuous round schema. The mechanism to deal with action decisions is supported by the Minds Interface that receives the current combat state for its analysis.

As the Combat Engine includes the mechanisms required by the learning algorithms, it provides the combat state information to the minds at the end of every action that it is involved.

Also, as the Combat Engine works it stores data of the combat into the database for the further information analysis and the representation by the GUI.

E. Mind Interface

The character controller, named **mind** in VBATTLE, selects actions on behalf of the actual character. In order to select the action the mind can take into account the state of the combat. Once the action is executed, and every time the state of the combat changes, the controller gets a feedback from the engine updating combat information. Minds are designed as flexible implementations, based on messages, that support the inclusion of different techniques implemented on different languages.

Moreover, the mind could also request the engine to include some information in the database as result of the operations taken on the mind to recover it on future instants. This operation is supported by particular engine services for information persistence. This service also considers the appropriate identification of this stored information to be properly retrieved by the mind when requested.

The desired interaction between minds and the engine is accomplished through the services offered by the message broker system of ActiveMQ from Apache.org that enables the interconnection of different elements implemented on wide range of languages.

This plug-in system to design minds support the creation of different implementations of control algorithms to decide the action in this game framework. This is the main goal for the creation of this framework, the possibility of testing different decision algorithms in a "more realistic" computer game (at least with the flavor of a computer war game).

F. Technological Issues

The technology used for this project is centered on the Java programming language. First, the Combat Engine and the structure of the code is modeled with Java classes. The communication between the engine and the minds, (or even the interface) is treated as events, these events are added to the round flow controller, logging it on the database if necessary.

The use of the ActiveMQ message broker provides loosely-coupled interaction between the pluggable elements such as minds or even the GUI and analysis tools. When the engine is running it could process the requests received through the broker and send the desired result back to the sender.

Although the message schema is not optimal for real-time interaction, the main objective of the VBATTLE framework is not the human interaction. Thus, this drawback is acceptable compared with the flexibility of creation of independent tools or controllers for the engine.

The logging is made both on the database and using log files (if the database schema is not flexible enough). The creation of new logging file format by the classes provided by the framework encourages the inclusion of personalized analysis mechanisms.

The realization of the GUI could be implemented, using the message schema, in nearly any platform, for instance: following the specifications of the jMonkey game engine

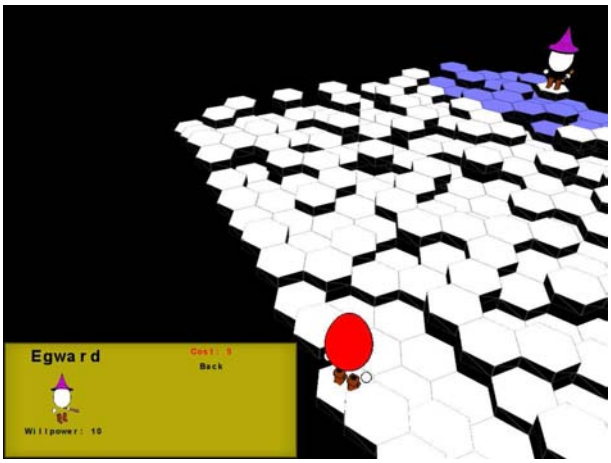

Fig. 3. Hard Boild Screenshot.

and presenting the appearance of a 3D hexagonal map displaying the character on the game board and it will have the capabilities of small interaction with the elements and visual logging elements. A preliminary design uses the ideas of Hard Boild[3] (see Figure 3); or using SDL library and working with a C++ hexagonal GUI or even Blender and Phyton engine.

The analysis of the results can be done accessing to the result database or logging files and it is completely decouple from the game engine providing the capability of creations of different analysis mechanism.

IV. Computational Intelligence Application

The main goal of vBattle framework is to provide a solid infrastructure to research and validate intelligence algorithms for controlling both combatants and group of combatants.

Every single simulation is called a **battle**, and it represents a sequence of events (derived from the actions selected by the characters' controllers) which last up to a given condition is met. In most of the cases this condition is when one of the sides is defeated, but it could be extended to support other alternatives: (1) simulated time limit, (2) wall-clock execution limit, or (3) a given circumstance, such as a particular combatant reaches a given position or performs a specific action.

The group of combatants participating in a battle as a same group is called a **faction**. The combatants include the character parameters (also the actions) as well as the controller schema (combatant minds).

A **tournament** is a set of factions that fight along a series of battles. On each of these battles, all or part of the factions are involved.

The tournaments are the mechanisms to explore computational intelligence algorithms. The tournaments can be restricted in order to reduce and focus the aspects under consideration:

- **one on one without scenario**: The combatants are already engaged in combat and the algorithms have to decide the best sequence of actions depending on the internal counter (HPs and EP) as well as opponent counters and the event sequence.
- **one on one with scenario**: Similar to previous one, but in this case the combatants are not engaged in combat, and they have to move across the scenario, chasing the opponent. The characters have to deal with path identification, scenario modifiers and the characteristics of its own actions.
- **group combat (kill'em all mode)**: Variants of the two previous cases but factions include multiple combatants or there could be more than two factions. Alliance and coordination aspects are important in this case, encouraging the construction or group-based tactics.
- **group combat (take the flag)**: Multiple factions with multiple combatants per faction with an objective different from killing the rest of the factions. In some cases, it is to reach and maintain a particular position ("*take the flag*") but can be expressed as different conditions. This tournament is a good testbed to consider charater's personalities (giving different reward functions depending on the specific interest of each combatant, such as protect the leader or survive).

Together with these general tournament scheme other aspects are considered:

- Pool of Combatants: In order to generalize strategies and to avoid overfitting in the construction of these strategies.
- Learning process: Controllers learn during tournament, storing intermediate information and improving their expected performance. vBattle provides the mechanism to record controller variables along different battles.

V. Preliminary Experiments and Results

As an example of the use of vBattle framework to study different computational intelligence techniques, the following scenario has been proposed:

- The objective is to compare the performance of different reinforcement learning algorithms for stochastic games, applying them to control characters in a fighting game.
- A one-to-one single character combat mode has been selected. In this mode, both combatants are engaged in meleé combat, with no modifiers or effects from the scenario. The objective of each character to defeat the opponent is to reduce its HPs to zero with a maximum limit of simulated time (simulations ticks).
- Three different reinforcement learning algorithms have been considered: PHC, WoLF and TERSQ [5].
- The scenario considers two different character profiles, thus six different contenders are configured (2 character types times 3 control algorithms).

A. Reinforcement Learning Algorithms

Three reinforcement learning algorithms have bee proposed for this study: PHC, WoLF and TERSQ

[3]http://code.google.com/p/hardboild/

1) Policy Hill Climbing (PHC): Policy Hill Climbing (PHC) was proposed in [6] as an extension to the Q-learning algorithm. Q-values are maintained as usually but, in addition, the algorithm also maintains the current mixed policy. This policy controls the probability to select a given action during the learning phase. It is updated by increasing the probability to select the best performing action according to a given learning rate.

2) Win or Learn Fast (WoLF): Win or Learn Fast (WoLF) policy was proposed also by [6] as an extension of the PHC policy reviewed in the previous section. The basic idea is to dynamically modify the learning rate used to encourage convergence without sacrificing rationality. Intuitively, the algorithm tries to learn quickly when it is losing and more slowly when it is winning. To determine if the algorithm is winning or losing, the current policy's payoff is compared with that of the average policy over time. For this purpose, the algorithm requires two learning rate parameters, one that will be used when the algorithm is losing, and a second that will be used when the algorithm is winning. This second learning rate is actually smaller than the previous one, in order to force learning when losing.

3) Tentative Exploration by Restricted Stochastic Quota (TERSQ): In [5] the Tentative Exploration by Restricted Stochastic Quota (TERSQ) algorithm was introduced. The main idea of this algorithm is to use a global stochastic quota in order to select the action to be executed. A binomial decision process is performed in such a way that actions with best Q-values are selected with the probability assigned by this quota, otherwise the rest of the actions are stochastically selected with a probability according to their Q-value ranking. The quota value is selected for each round based on three different criteria. From these criteria, three phases can be established: (1) *Tentative Phase* in which the algorithm tries all the possible quota values (from a finite set of values, named) to get an initial estimation of the performance of every possible quota, (2) *Adjustment Phase* where quota values are proportionally chosen according to their average performance (which is updated at the end of each round), and (3) *Optimal Quota Phase* where the quota value with highest average performance is selected for the rest of the learning process. The usual Q-learning technique is applied during all the process, learning while quota value is also computed.

B. Experimental Results

To evaluate these learning algorithms on this environment, two different character profiles, A and B, have been created. Each profile defines specific HPs, EPs and action characteristics. For each of these profiles, the three RL algorithms are used (TERSQ, PHC and WoLF) resulting six different characters (each of the two profiles and each of the three RL algorithms). The experiment consists of 10 series of 200000 battles. For each battle, two characters are randomly selected from the six available characters. The Q-values and the learning rates are reseted when each completed series begins.

TABLE I
RATIO OF WINS VERSUS OTHER CHARACTERS

	Wins					
	$TERSQ_A$	PHC_A	$WoLF_A$	$TERSQ_B$	PHC_B	$WoLF_B$
$TERSQ_A$		50,56%	50,44%	44,61%	39,32%	38,38%
PHC_A	49,44%		50,18%	36,49%	34,91%	33,32%
$WoLF_A$	49,56%	49,82%		35,87%	34,57%	33,13%
$TERSQ_B$	55,39%	63,51%	64,13%		53,94%	53,02%
PHC_B	60,68%	65,09%	65,43%	46,06%		49,42%
$WoLF_B$	61,62%	66,68%	66,87%	46,98%	50,58%	
Wins	55,33%	59,19%	59,45%	41,96%	42,55%	41,34%

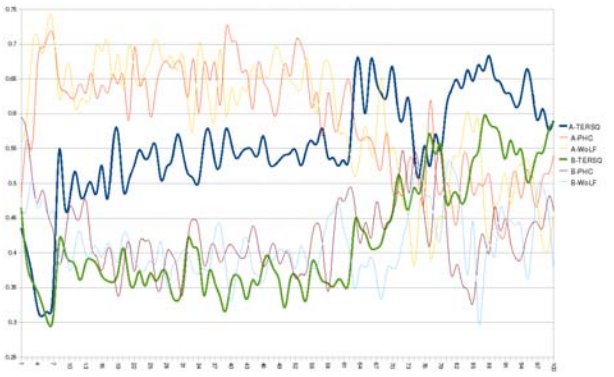

Fig. 4. Winig ratio evolution of PHC, WoLF and TERSQ.

Table I presents the wining percentage for each pair of characters averaged for the 10 executions. These results show a better performance of PHC or WoLF for each of the character profiles.

Table II shows the wining ratios restricted to the last combats of every experiment, once quota value has been selected. These last results emphasize a significative better performance of the two characters controlled by TERSQ algorithm. The two character profiles are different: **B** is worse than **A**, but despite of this B-TERSQ character profile beats nearly half of times against **A**-profiles, outperforming others **B**-profiles.

The figure 4 shows the evolution of the winnig ratio along the combats with the inflexion points that marks the different phases at 2000 (end of Tentative) and 120000 (end of Quota Adjustment).

TABLE II
RESULTS ON FIXED σ STAGE

	Wins					
	$TERSQ_A$	PHC_A	$WoLF_A$	$TERSQ_B$	PHC_B	$WoLF_B$
$TERSQ_A$		37,38%	36,52%	52,23%	33,96%	33,05%
PHC_A	62,62%		49,47%	49,07%	41,62%	40,52%
$WoLF_A$	63,48%	50,53%		49,91%	42,93%	41,58%
$TERSQ_B$	47,77%	50,93%	50,09%		51,71%	48,71%
PHC_B	66,04%	58,38%	57,07%	48,29%		48,98%
$WoLF_B$	66,95%	59,48%	58,42%	51,29%	51,02%	
Wins	61,56%	51,53%	50,47%	50,13%	44,27%	42,61%

VI. FUTURE WORK AND CONCLUSIONS

vBATTLE is currently under development and it is intended to be available on the next year, but the initial result are promising and it could provide an interesting workbench for further researches.

The evaluation of different learning algorithms validates this framework as an interesting environment for the artificial intelligence in game programming. The use of different level of detail in the same engine enables the research of multi layered techiniques in a fully controllable game engine. The possibility of off-line simulations creates a perfect frame for the use of techniques more complex and computationally massive. The future development of vBATTLE could create a new competition framework for techniques with a good modularity for the interaction in different types of tournaments.

vBATTLE framework is under development and only basic features are fully integrated right now. A first public use release is planned for the second half of the next year.

ACKNOWLEDGMENTS

The authors would like to thank the Spanish Ministry of Science (TIN2007- 67148 and TIN2006-14630-C03-02). And also with the Madrid Regional Education Ministry IV PRICT

REFERENCES

[1] P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma, "Online adaptation of game opponent ai in simulation and in practice," in *4th International Conference on Intelligent Games and Simulation (GAME-ON 2003)*, 2003, pp. 93–100.

[2] V. Corruble, C. Madeira, and G. Ramalho, "Steps toward building a good ai for complex wargame-type simulation games," in *Proceedings of The 3rd International Conference on Intelligent Games and Simulation, London, United Kingdom*, 2002.

[3] M. Buro and T. Furtak, "On the development of a free rts game engine," in *GameOn'NA Conference*, 2005.

[4] K. O. Stanley, I. Karpov, R. Miikkulainen, and A. Gold, "Real-time interactive learning in the nero video game," in *AAAI*. AAAI Press, 2006.

[5] L. Peña, A. LaTorre, J.-M. Peña, and S. Ossowski, "Tentative exploration on reinforcement learning algorithms for stochastic rewards," in *HAIS*, 2009, pp. 336–343.

[6] M. Bowling and M. Veloso, "Rational and convergent learning in stochastic games," in *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI)*, August 2001, pp. 1021–1026.