A Method for Generating Emergent Behaviors using Machine Learning to Strategy Games

Alex F. V. Machado Esteban W. Clua Bianca Zadrozny

Universidade Federal Fluminense, Instituto de Computação. Niterói/RJ, Brasil

Figure 1: Darwin Kombat, strategy game simulator developed for the Machine Learning experiments.

Abstract

This work proposes the use of machine learning for the creation of a basic library of experiences, which will be used for the generation of emergent behaviors for characters in a strategy game. In order to create a high diversification of the agents' story elements, the characteristics of the agents are manipulated based on their adaptation to the environment and interaction with enemies. We start by defining important requirements that should be observed when modeling the instances. Then, we propose a new architecture paradigm and suggest what would be the most appropriate classification algorithm for this architecture. Results are obtained with an implementation of a prototype strategy game, called Darwin Kombat, which validated the definition of the best classifier.

Keywords: Machine learning, emergent systems, strategy games.

Authors' contact:
alexcataguases@hotmail.com
{esteban,bianca}@ic.uff.br

1. Introduction

Machine learning studies the development of techniques and algorithms that allow computers to learn how to perform tasks based on empirical data. One of the most commonly studied tasks in machine learning is classification, which consists in mapping an instance represented by a set of attributes into a class. The learning is usually done using as input a database of labeled instances also called an instance base [15].

For digital games there are many benefits and reasons for using these techniques, such as adapting the game to player actions or determining solutions for problems that are difficult to predict in the development process. As application examples, we can cite: the adaptation of a computer character strategy when playing against the human player character based on its actions in a fight or shooting game; the definition of the best profile for a football player that will be acquired by a trainer which is computer-controlled, so that the team performance can be in improved in a management simulation game; the adaptation of the race strategy of a vehicle to an unknown race circuit so as to minimize fuel usage, time and accident risk in a racer style game; among others.

In digital games, the learning process can be done in two different ways, based on adaptive behaviors [10]: online and offline learning. The online approach shows more efficiency, since it implies in deciding how the behavior of an agent must change, in response to captured information from the world. The offline solution is applied in more specific and well delimited environments.

Among the most commonly used machine learning techniques, we can highlight neural networks, genetic algorithms, Bayesian algorithms and decision trees. Typically game engines do not include implementations of such techniques, suggesting that developers should use separate software packages. For solving video game problems, there are few solutions that incorporate the requirements usually demanded. One of these packages is WEKA [17], which includes many different machine learning algorithms

In general, the use of machine learning in digital game development is either avoided or reduced because it is considered of high risk. However this is probably due to the fact that neural networks, which is a hard to deploy technology, is the usual choice [16].

An exceptional commercial case that uses classification approaches based on machine learning strategies is the game Black and White [8][10]. In this game, the data modeling is made based on instantiations and learning rules are generated from the induction of decision trees. An indirect adaptation, classified as online learning, is done for determining which objects are appropriated for feeding a creature. With this strategy, this character is induced to choose only the options that are good for his knowledge base.

Another important reference is the game Treasure Hunt [14],which uses machine learning, decision trees and Naive Bayes, applied to an NPC that will help the player on his mission objectives achievements.

The main contribution and purpose of this work consists on modeling the problems that use instance bases focused on the classification problem, which differs from solutions that are usually presented [1][16], often related to neural network or genetic algorithms approaches.

2. C4.5, Naive Bayes and M5P

Classifiers usually take as input a data set, and generate as output a mathematical expression, rule set or a decision tree that embodies the knowledge contained within the data. The knowledge extracted can be used either directly by the system or it can be applied to new instances, not classified yet. Among many different existing classifier learning methods, we chose to work with the following:

- C4.5 decision tree learner, since it is one of the most traditional learning algorithms based on decision trees [15];
- Naïve Bayes, since it is one of the most practical and well known probabilistic methods, at least when it comes to computational efficiency [15];
- M5P decision tree learner, since it allows real numbers for the class, which turned out to be more appropriate for our problem.

Each one of the algorithms works in a different way. C4.5 infers a decision tree from the data, allowing the tree to grow until if fits the training data as well as possible. Then the algorithm prunes the decision tree by eliminating branches that are not expected to be useful for new examples [15].

Naive Bayes is a probabilistic learner which calculates the probability of each class given the values of the attributes, by assuming that the attributes are independent of each other given the class. By using this assumption, it can run very efficiently compared to other probabilistic learners and still give accurate results in many real problems [15].

M5P [8] is a reconstruction of Quinlan's M5 algorithm [7] which has been designed for the induction of trees of regression models. M5P generates a conventional decision tree with linear regression model at the leaves. Its main steps include the induction algorithm that is used to build the decision tree, the pruning which is done regressively from the leaves and the smoothing procedure that is used to avoid abrupt discontinuities between subtrees. This method generates models that are compact and relatively understandable.

The Naive Bayes algorithm produces probability tables as output, which can be incorporated into executing systems in real time. In a similar manner, C4.5 produces decision trees that can be translated into rules and also incorporated in real time. M5P can be seen as a hybrid model since it is a combination of decision tree with a linear function that allows it to predict real values.

3. Creating an Intelligent Agent with Learning for Games

A rational intelligent agent or simply intelligent agent is a system designed to achieve a certain goal and which is capable of perceiving its environment through sensors and acting upon its environment through actuators. Intelligent agents who can also learn, that is, change their behavior over time according to the results of their actions are known as Agents with Learning [6]. Learning makes agents autonomous in the sense that they are able to adapt to new situations for which they have not been explicitly designed.

Intelligent agents are present in different aspects of electronic game development, from the systems that calibrate game level according to user performance up to simple NPCs.

Modeling a system using the intelligent agent with learning architecture eases system development as whole. For this reason, here we have opted to use this architecture integrated with a machine learning module.

When defining the instance base for the machine learning module, we followed some pre-processing guidelines which are listed below [5]:

- Ordinal qualitative attributes should be represented as numerical attributes instead of categorical attributes. This allows the algorithm to take advantage of the value order, and as a consequence, less data will generally be needed to learn a concept satisfactorily. Example: the attribute water_temperature {cold, warm, hot, very hot} should be transformed to water_temperature_numeric, with values between 0 and 3.
- Restriction guarantees should be given to avoid inconsistent data. We prevent the insertion into the instance base of instances

that have inconsistent or out-of-bounds values. Example: the agent_points with value in 100 in a stage where it could reach at most 10 points.

- Aiming for dimensionality reduction. The instance base should only have the attributes that are deemed necessary for learning. This will make learning easier and it will also improve the running time, which is a requisite for real time processing systems like digital games.

4. A Development Framework

In this section we propose an architecture for the development of game applications with machine learning which makes use of free tools and components. This framework is especially tailored to digital games which are modeled used the intelligent agent paradigm.

4.1 Tecnology

This framework integrates the following technologies: Unity 3D (game engine), Microsoft Visual Studio (server development IDE), C#, Weka and IKVM.

Weka[17] is a free open source machine learning toolbox that is widely used among the developers in this área. It provides an API and a standalone jar file so that its algorithms can be used freely. The algorithms we use in this paper (C4.5, Naïve Bayes and M5P) have all been implemented within Weka. We note that the C4.5 algorithm in Weka is called J48.

Unity3D 2.x [11] is a game development engine which has the main components needed for designing games rapidly, such as a physics engine, collision systems, sound system and a high level programming language based on C, among others.

Unity 3D has an interface for programming in Mono. It incorporates key .NET components, including a compiler for the C# programming language and a complete suite of class libraries.

Since we opted for developing the game in Unity3D, it was necessary to convert the standalone Weka module into a DLL that would be interpreted by Mono's C#. For this integration, we used IKVM [2], which provided the DLL generation and served as reference for accessing the Java API from this DLL (for this purpose we have used a standalone module from IKVM as well).

Despite all the advantages of using Mono, the version included in the Unity3D package had incompatibilities with the Weka DLL interpreted by IKVM in the experiments we ran. More specifically, the problem was related to the garbage collector. To circumvent this problem, we opted for integrating Unity3D with a server module developed using Visual Studio and making the connections through sockets.

We chose the Microsoft Visual Studio .NET platform as the main server development tool, since it is a Rapid Application Development (RAD) tool and allows the interoperability between multiple programming languages [9]. Within this platform, we chose the C# programming language for incorporating Weka's DLL. C# offers all the resources of a modern programming language, including support for objects, interfaces, components and managed code.

4.2 System Architecture

For developing the agent architecture we gave special attention to agent modeling and reinforcement learning. This architecture was specially designed for the creation of an agent capable of generating strategies for a specific game (which shall be explained in section 5). Nonetheless, the same basic principles could be applied for the creation of different games.

As demonstrated in Figure 2, the agent in this application is represented by AI SYSTEM module and the environment is the UNITY3D GAME. We can characterize it as an intelligent agent because it has a sensor, the character score; a reasoner, with learning modules and an instance base; and an actuator, which generates a new character to be put into the environment.

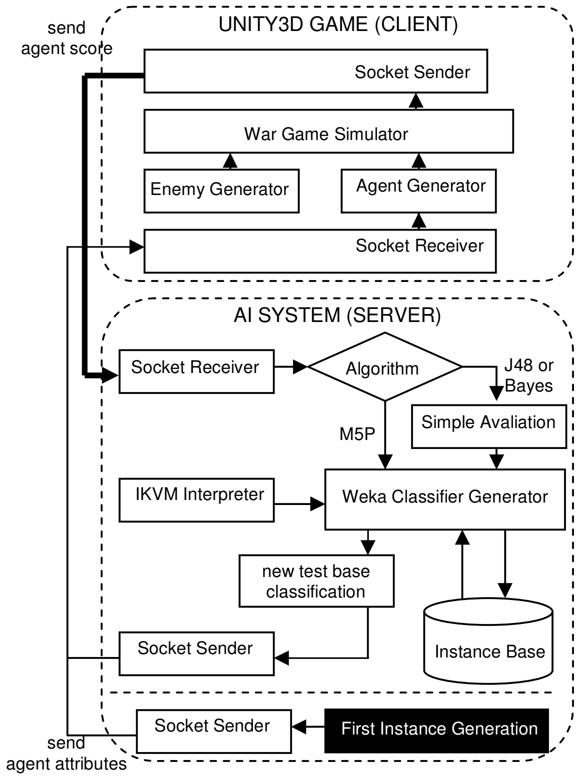

Figure 2. Intelligent agent architecture with learning applied to the generation of new strategies for a game.

The system starts in the IA SYSTEM module (First Instance Generation), generating a new random instance and sending it to the environment (UNITY3D GAME) through sockets. We note that the system uses on-line training and it does not have any classification rule for the first instances.

In the environment the instance is incorporated to the game graphical elements, evaluated and sent back to the agent (character score), also through sockets. From this point on, the process depends on the chosen classification algorithm:

- In the case of M5P, this score is kept in the instance base and serves as the base for generating the classification function;
- In the case of Naïve Bayes or J48, the score value is transformed in to a discrete value by testing whether the value is above or below the average. The values above the average of all scores are considered "good", whereas values below the average are considered "bad". This new label is inserted into the instance base so that we can run the classifiers using them. This step is necessary because the Naïve Bayes and J48 algorithms do not deal with continuous class values in the classification attribute.

Finally, the system generates a new individual pool, the classifiers are used to choose good individuals and they are sent to the environment, restarting the cycle. If M5P is used, the system generates 10 times as many individuals as needed and the new test base classification module chooses the best ones. For J48 and Naïve Bayes, since there is no ranking from best to worst (only a discrete "good" and "bad" classification), we generate random individuals and select only the ones with "good" classification.

Although we use classification algorithms, which are usually used in a supervised learning setting, this system can be seen as implementing a kind of reinforcement learning strategy because classification is applied repeatedly with the goal of improving an agent generation strategy without direct supervision. The classifier represents an agent generation strategy, and in each round, it is used to generate agents. The agents are rewarded with scores (rewards in reinforcement learning) and the best ones are used in the instance based for creating the classifier for the next round. Since the environment is dynamic, this allows the adaptation of the agent strategy to the environment, thus improving the generation strategy.

5. A model for Machine learning applied for strategy games

This session will present the implementation of a strategy game using our proposed architecture. This application will be used to evaluate classifiers in different situations.

5.1 Game concepts

The prototype developed is called Darwin Kombat (Figure 1) and can be classified as an action turn based strategy game. This experimental version implements only the simulation of simple agent reactions and the agent generation with learning process.

The game starts with a battle environment (Figure 3), where two rival groups fight one against the other with the objective of eliminating all their opponents. Each group may have from 1 up to 40 components. Each participant, represented by an agent, has a contact weapon (sword) and a launching weapon (shot).

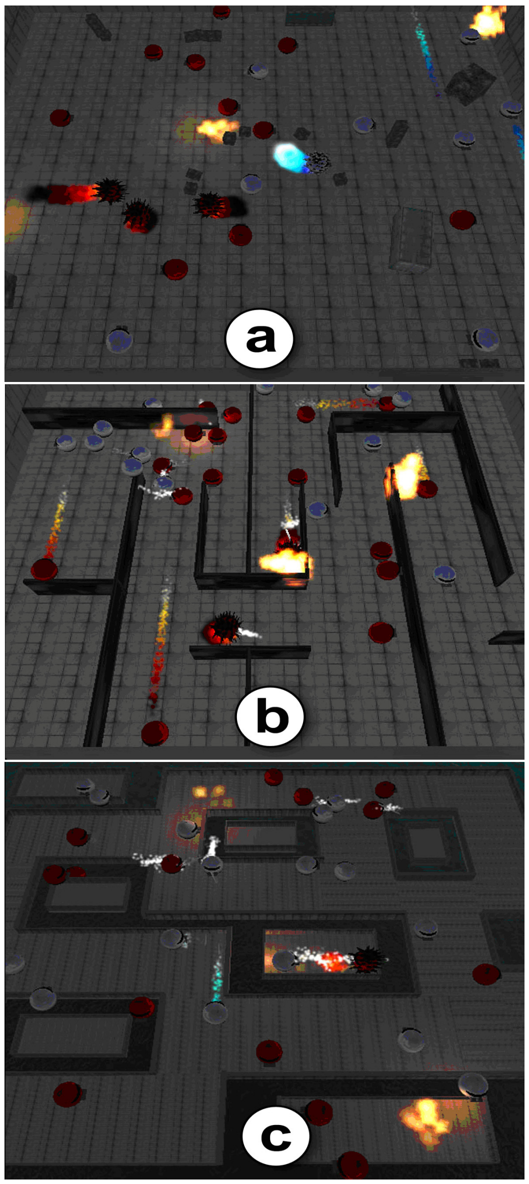

Figure 3 – The three different tests environments: (a) Zone-1, (b) Mazze Zone, (c) Sea Zone. The first group of tests where realized in Zone-1.

Each agent has a technical specification with the following attributes: w_speed, move velocity; w_shot, shoot velocity; w_life, resistance; w_sword, chance of eliminating the enemy with a sword movement (when close to the enemy); and w_delta-S, the space that is capable to move before an random turn around. Each one of this attributes may change from 1 (low) to 5 (high).

A Delta-S high does not yield necessarily a good character. For this reason it is not used in calculating the limit of the profile. This opens scope for the creation of different strategies (next sections).

Each agent does not sense the presence of its enemies or friends, so it is not capable to detect an agent and then turn around in order to shoot on it. This is not possible even when this opponent is close to it. Despite this shot of an agent does not affect your ally.

There may be many different types of battle environments (Figure 3 a, b and c). The simplest one is called Zone-1, which has few obstacles in its paths and no internal walls. The second scene is called Mazze Zone and is a labyrinth that has high walls, not allowing long shoots and for this reason creating a set of small combat cells. The third and last environment is called Sea Zone, that is an island on the sea. Although this scene limits agents movements, itallows targetting a shoot at any other environment agent and depending on the situation may also throw an enemy to the sea in case it is close to the border.

Moreover, in this last phase about 10% of the characters dies initially by falling directly into the sea (i.e. to increase the dynamics of the environment).

The battle is composed of two groups of agents: a blue one, with fixed attributes defined by the developer and a red one, with random attributes generated by the agent generator system, based on the learning machine definitions.

The game establishes a limit time for a battle. The winner will be the one that kills all the opponent agents or at least the largest number.

This prototype game represents a dynamic simulation system for the enemies and the environment and allows the generation of strategies for the teams. The strategy of technical data of the agents is a resource also used in different commercial games, such as Winning Eleven: Pro Evolution Soccer [3] and World of Warcraft [4].

5.2 Development Process

The 3D game itself was developed using the Unity game engine and the decision of the characteristics was done using .Net. The communication process was developed using sockets, which did not generate problems related to waiting time.

Given the option of validating on each frame the last generated element or all of them, preliminary experiments demonstrated that the largest the entrance instance set, the better the classification process performance (victories of the learning agent). For this reason we chose to validate the entire instance set.

5.3 Experiments

We performed 30 different tests with each of the three algorithms. The number of characters for each launched team was 20, being this amount defined in order to be big enough for a fast instantiation accumulation for each cycle and small enough for allowing a correct analysis of the evolution of the established algorithm strategy. Each test was composed of 10 phases, being the characters generated at the first one with randomized characteristics and in the following phases these attributes were regulated by the classifiers.

The enemies (blue team) were configured with the following technical characteristics: w_speed = 4, w_life = 4, w_shot = 4, w_sword = 3 and w_delta-S = 3, where 1 is the lowest value and 5 the highest value.
The characters configuration generated by the intelligent system totalized always 10 points, with out the Delta-S.

5.4 Comparison between the Classifiers

In the experiment conducted by McQuiggan and Lester [14] on evaluation of empathy between characters in a game (RPG focused in the field of social networking), we observed that the naive Bayes needed less instances to acquire the knowledge in comparison with the decision tree. In our experiment, the naive Bayes obtained the same advantage over the classifier based on decision tree (J48).

As can be seen in Figure 4, the y-axis represents the number of dead enemies and its x-axis stage. Vertical bars represent the standard deviation of the statistical tests. The number of killed red agents was disregarded in the graphs because in most cases was inversely proportional to the number of enemies killed, representing redundant information.

An unquestioned analysis is that both the algorithms were successful: they learned how to eliminate the opposing team. It is observed in ancestry lines from the first test.

J48 has obtained the worst performance with the largest standard deviation. The Naive Bayes proved to be better than the first with the shunting line tending to 0. Despite this, M5P was the best of the three algorithms.

This behavior is due to its numerical classification, which guarantees the least loss of information (so accurately) for the next generations (stages). While the first two algorithms just map the set of attribute values to a categorical target value.

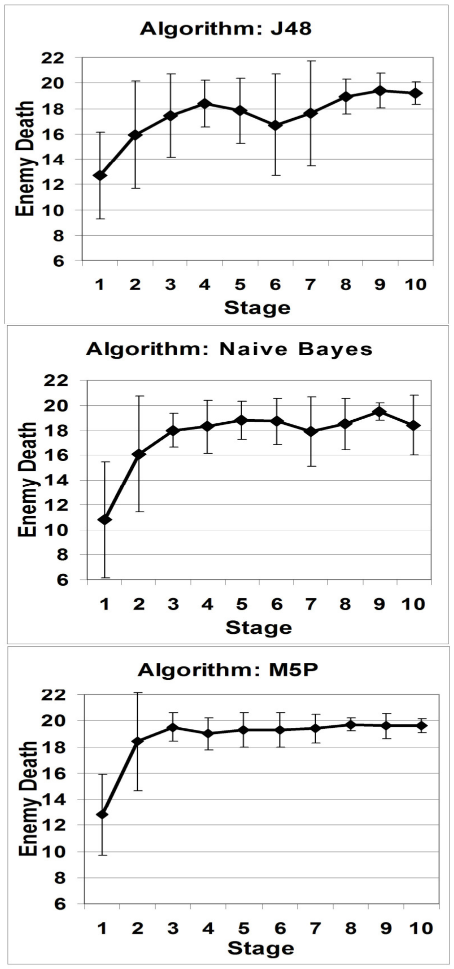

Figure 4. Graphic evolution of J48, Naive Bayes e M5P algorithms.

A very important piece of information that it is not shown in the graphs arethe generated strategies (detailed in next subsection). The J48 algorithm on each test was able to generate a strategy and evolved it. For example, in stage three, it produced the rule that characters have to tend to be fast and high life and maintained this strategy (or some derivation thereof) until stage 10. Naive Bayes, due to its characteristic of assuming the independence of attributes, only emphasizes the good attributes without considering their relationship with other attributes. Causing best battalions generation (in comparison with J48) although to possess personages with diversified characteristics (and diversified strategy). The M5P showed characteristics of both. In practice for the games, depending on the game, this factor may have great relevance since the developer will have to decide between making an algorithm that "wins the game" or that "keeps a team pattern characteristic", for example.

5.5 Emerging Behaviors Analysis

Through the games developed by Crocomo et al. in [12] and [13] it has been demonstrated that machine learning in games (in this case, evolutionary algorithms) are capable of allowing strategies to emerge from simple behaviors.

Through the resource management system for generating strategies present in Darwin Kombat, we can show that the strategy is able to adapt to the enemy and environment.

Let us consider the tree generated in the fourth phase of one of the J48 in the previous experiment, which ensured a victory with 20 enemies killed:

```
w_shot <= 3: amateur
w_shot >  3
    |     w_sword <=1: professional
    |     w_sword > 1: amateur
```

It indicates that the new generation of characters must have high w_shot and low w_sword, which indicates a new generation of individuals who have a very high firepower, speed and life therefore tend to be high.

A new group of tests based on 5.3 parameters subsection was conducted to demonstrate environment adaptation. It was applied to the Mazze Zone and Sea Zone stages and only with the M5P algorithm and obtained the following result (Figure 5):

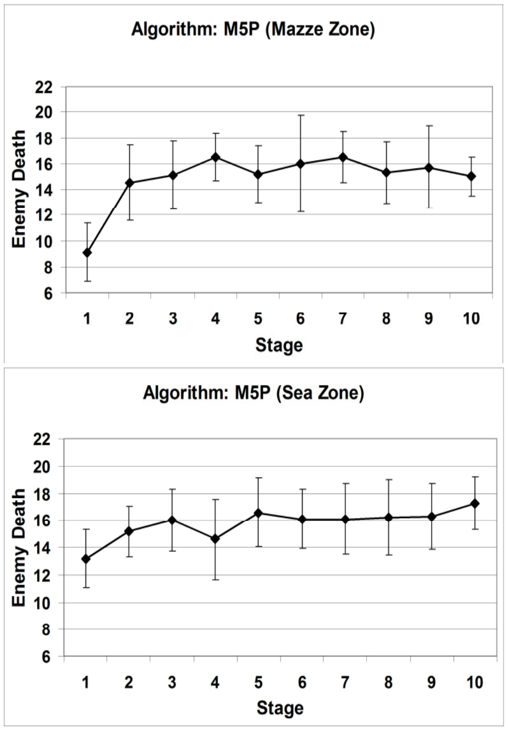

Figure 5. Graphic evolution of M5P algorithm in two other environments.

Although these two new tests have even more dynamic components, M5P demonstrated evolution / adaptation. Some models generated by the algorithm (in tests with victory) can be checked below:

```
w_classificacao =
-0.43*w_life -0.40*w_shot +
0.5*w_deltaS +1.54
```

This example indicates that the generator has stipulated that the attributes w_life and w_shot must be low and the w_delta-S high. This means that other attributes (w_sword and w_speed) will be high and, due to the w_delta-S, the character will walk far before turning randomly. It is concluded that “ninja” was created one, a character who does not shoot and chase the enemy. Appropriate for this stage of labyrinths, in which the walls hinder the efficiency of release (w_shot). In Sea Zone we could observe this other model:

```
w_classificacao =
-0.20*w_speed -0.33*w_sword -
0.31*w_deltaS +1.43
```

This example shows the reverse: w_shot, w_life and w_delta-S must be high. This means that it is not good at move fast (as it may fall into the sea) and it is more efficiently in eliminating enemies. This created, therefore, a character which behaves like a "tank-of-war".

6. Conclusion

This paper presents a study of emergency in strategy games through the use of classical classifiers related to machine learning.
From this study, we define key requirements to be observed for use of learning in games, as system modeling in intelligent agent form and some care in the database instances preprocessing to ensure performance in real-time processing.

We define a free development framework based on an architecture centered on the tools Unity3D, MS Visual Studio and Weka. On this architecture we implemented the game Kombat Darwin and observed the sprouting of emerging strategies. Through experiments we demonstrate the advantages of using a classifier with ordinal quantitative return (the M5P) over its competitors based on qualitative attributes classifiers (J48 and the Naive Bayes).

Acknowledgements

This work has been partially funded by Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais.

References

[1] BUGAJSKA, M.D., SCHULTZ, A.C., TRAFTON, J.G., TAYLOR, M., MINTZ, F.E., 2002. A hybrid cognitive-reactive multi-agent controller. In: Proceedings of the IEEE/RSJ Intl. Conference on Intelligent Robots and Systems, EPFL, Lausanne, Switzerland.
[2] IKVM.NET, http://www.ikvm.net/
[3] WIKIPEDIA: Pro Evolution Soccer (series). At http://en.wikipedia.org/wiki/Winning_Eleven
[4] WIKIPEDIA: Warcraft. At http://en.wikipedia.org/wiki/Warcraft
[5] TAN, N., STEINBACH, M., KUMAR, V., 2006. Introduction to Data Mining, Addison-Wesley.
[6] RUSSELL, S. J., NORVIG, P., 2003. Artificial Intelligence: A Modern Approach (2nd ed.), Upper Saddle River, New Jersey: Prentice Hall, ISBN 0137903952.
[7] QUINLAN, R. J., 1992. Learning with Continuous Classes. In: 5th Australian Joint Conference on Artificial Intelligence, Singapore, 343-348.
[8] WANG ,Y., WITTEN, I. H., 1997. Induction of model trees for predicting continuous classes. In: Poster papers of the 9th European Conference on Machine Learning.
[9] LIBERTY, J., 2001. Programming C#. O’Reilly.
[10] HOULETTE, R., 2003. Player Modeling for Adaptive Games: AI Programming Wisdom 2, p. 557. ISBN 1584502894.
[11] UNITY TECHNOLOGIES: Unity 3D User Manual.
At www.unity3d.com/support/documentation/Manual
[12] CROCOMO, M. K., SIMÕES, E. V., 2008. Um Algoritmo Evolutivo para Aprendizado On-line em Jogos Eletrônicos. SBGames.
[13] CROCOMO, M. K., MIAZAKI, M. Simões, E. V., 2007. Algoritmos Evolutivos para a produção de NPCs com Comportamentos Adaptativos. SBGames.
[14] MCQUIGGAN, S. W., LESTER, J. C., 2007. Modeling and evaluating empathy in embodied companion agents. In: International Journal of Human-Computer Studies, v.65 n.4, p.348-360.
[15] MITCHELL, T. M., 1997. Machine Learning, McGraw-Hill.
[16] CARVALHO, F. G., 2004. Comportamento em Grupo de Personagens do Tipo Black&White. Dissertation. PUC-Rio – Digital Certification N° 0210488/CA.
[17] HOLMES, G., DONKIN, A., WITTEN, I.H., 1994. Weka: a machine learning workbench. In: Proceedings of the 1994. Second Australian and New Zealand Conference on Intelligent Information Systems, Brisbane, Australia, pp. 357-361.