Learning Virtual Agents for Decision-Making in Business Simulators

Javier García and Fernando Fernández
Universidad Carlos III de Madrid
Avenida de la Universidad, 30, 28911
Leganés, Madrid, Spain
Email: fjgpolo,ffernand@inf.uc3m.es

Fernando Borrajo
Universidad Autónoma de Madrid
Crta. de Colmenar Viejo, km. 14, 28049, Madrid, Spain
Email: fernando.borrajo@uam.es

***Abstract*—In this paper we describe SIMBA, a simulator for business administration, as a Multi-Agent platform for the design, implementation and evaluation of virtual agents. SIMBA creates a complex competitive environment in which intelligent agents play the role of business decision makers. An important issue of SIMBA architecture is that humans can interact with virtual agents. Decision making in SIMBA is a challenge, since it requires handling large and continuous state and action spaces. In this paper, we propose to tackle this problem using Reinforcement Learning (RL) and K-Nearest Neighbors (KNN) approaches. RL requires the use of generalization techniques to be applied in large state and action spaces. We present different combinations in the choice of the generalization method based on Vector Quantization (VQ) and CMAC. We demonstrate that learning agents are very competitive, and they can outperform human expert decision strategies from business literature.**

I. Introduction

Business simulators are a promising tool for research. The main characteristic of SIMBA (*SIMulator for Business Administration*) [2] is that it emulates business reality. It can be used from a competitive point of view, since different companies compete among themselves to improve their results. In this paper, SIMBA is considered as a multi-agent framework where the different agents manage their companies in different ways. SIMBA can include several autonomous agents to play the role of competing teams and, based on the research on decision making patterns of human teams, further research is made to improve the complexity and effectiveness of such intelligent agents.

Decision making in SIMBA requires handling more than 100 continuous state variables, and more than 10 continuous decision variables, which makes the problem hard even for business administration experts. The motivation of this paper is the design, implementation and evaluation of virtual agents in SIMBA using different machine learning (ML) approaches. The goal is that the developed agents can outperform human-like behavior when competing against hand-coded and random virtual agents, but also against expert humans players.

Human players have experimented the consequences of their decisions in competition with the developed virtual agents. But, given that the agents try to "win" in all cases, they make the game too hard for novice players. So "pedagogical" objectives for human players competing with our virtual agents, are not directly included in the goal of this paper. Designing virtual agents whose behavior challenges human players adequately is a key issue in computer games development [23]. Games are boring when they are too easy and frustrating when they are too hard [8]. Difficulty of the game is critically important for its "pedagogical" worth. The game difficulty must be such that it is "just barely too difficult" for the subject. If the game is too easy or too hard, "pedagogical" worth appears to be less efficient. So most games allow human players adjust basic difficulty (easy, medium, hard).

However, developing agents that can outperform human-like behavior, under narrow circumstances, can do pretty well [15] (ex: *chess* and *Deep Blue* or *Othello* and *Logistello*). *Deep Blue* defeated World Chess Champion Garry Kasparov in an exhibition match. Campbell and Hsu describe the architecture and implementation of their chess machine in the paper [7]. A few months after this chess success, *Othello* became the new game to fall to computers when Michael Buro's program *Logistello* defeated the World Othello Champion Takeshi Murakami. In the paper [3], Buro discusses the learning algorithms used in his program. Thus, the goal of this paper is the development of virtual "business" agents that can be able to beat hand-coded and random virtual agents, but also human business experts.

To do so, we use two different learning approaches. The first one is Instance Based Learning (IBL). In this paper we propose the Adaptive KNN algorithm, a variation of KNN, where experience tuples are stored and selected automatically to generate new behaviors.

However, decision making for business administration is an episodic task where decisions are sequentially taken. Therefore we also propose to use Reinforcement Learning (RL). The RL agents developed need to apply generalization techniques to perform the learning process, given that both the state and action spaces are continuous. In this paper, we propose two different generalization methods in order to tackle the large state and action spaces. The first one, Extended Vector Quantization for Q-Learning, uses Vector Quantization (VQ) to discretize both the state and action spaces, extending previous works where VQ was used only to discretize the state space[6]. Some tasks have been solved by coarsely discretizing the action variables [14], but up to our knowledge, this is the first time that VQ is used to discretize the action space.

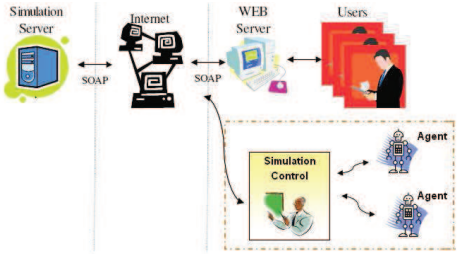

Fig. 1. SIMBA's Arquitecture

The second generalization approach, CMAC-VQQL, is based on the combination of VQ to discretize the action space and CMAC (*Cerebellar Model Articulation Controller*) [1], which is motivated by CMAC's demonstrated capability to generalize the state space.

Section II describes SIMBA. Section III introduces the learning approaches proposed, while Section IV shows how these approaches have been used to learn the virtual agents for decision making in SIMBA. Section V shows comparative results of the virtual agents, when competing among them but also when competing against expert human players. Section VI summarizes the related work. Last, Section VII concludes.

II. SIMBA

In this section, *SIMBA* simulator is described in detail.

A. SIMBA's Architecture

Figure 1 shows the architecture of the business simulator from a Multi-Agent perspective. The architecture designed enables multiple players to interact with the simulator, including both software agents and human players. The main components of the system are:

- **Simulation Server**: Once all decisions are taken for the current round, it computes the values of the variables in the marketplace for every player. Finally, it sends the results computed to each player. The player (software of human) uses these results to choose the best decisions in the next round of the simulation.
- **Simulation Control**: It manages the software agents and their decisions. It receives the decision taken by the software agents and sends them to the Simulation Server. The simulation server the results computed to the simulation control. The simulation control sends the results to the corresponding software agent.
- **Software Agents**: They represent an alternative to human players. In every step, the software agents receive the results computed for the Simulation Server. The software agents use this information to take the decisions for the next round of the simulation.

B. Business Human Strategies

Different business strategies appear in the business literature, and they all could be followed to manage the companies in SIMBA, as will be shown in Section V. We describe some classical ones:

1. Incremental decisions. This type of business strategy is based on incremental decisions for all decision variables, which typically ranges from a 10% to a 20%. This business strategy is considered as a conservative behavior.

2. Risk decisions. It is based on strong changes in business decisions. It has strong impacts in market reactions, and is useful to detect gaps and market opportunities.

3. Reactive. An organization with this type of strategy attempts to locate and maintain a secure niche in a relatively stable product or service area [11].

4. Low cost strategy. With this strategy, managers try to gain a competitive advantage by focusing the energy of all the departments on driving the organization's costs down below the costs of its rivals [12].

5. Differentiation and specialization. A differentiation strategy is seen when a company offers a service or product that is perceived as unique or distinctive from its competitors [12].

Which strategy management is chosen in every moment depends on the organization's strengths and its competitor's weaknesses.

C. Autonomous Decision Making in Simba

The goal of this section is to describe how a *SIMBA* software agent can be implemented. To do this, we describe the state and action spaces, the transition function to transit between states and the variable to maximize.

State Space. The state computed in every round or simulation step is composed of 174 continuous variables. Table I shows some of the features that compose the state space.

Action Space. The players (software or humans) must approach the decisions on the different functional areas of their companies. Each market in the competition requires the use of 25 variables. This is an indicator of *SIMBAS's* capacity to approach the complexity of managerial decision-making. In our experiments, we consider a subspace of the total action space and we use only the ten variables shown in table I. This reduction was suggested by the experts, because the discarded variables are not very significant. All the actions that the agents can perform are constrained by the semantic of the business model. For instance, a company can not sell its product if it does not have stock.

Transition function. The different players participate in a simulation in a step by step round mode. Each simulation step is called a period, which is equivalent to three real months. When a round ends, the time machine is run. By doing this, the simulator integrates the previous periods situation, the teams' decisions, and the parameters of the general economic environment together with those of each geographic market, and orders the Simulators Server to generate output information for the new period.

Variable to maximize. The agents try to maximize the result of the exercise (profit). From a RL point of view, the objective is to maximize the total reward received. In this case, we

TABLE I
A SUBSET OF FEATURES OF THE STATE AND ACTION SPACES.

FEATURES of the State Space	**FEATURES of the Action Space**
Account value	Selling price
Human resources	Advertising expenses
Material cost	Network sales budget
Operating margin	Commercial information
Financial expenses	Training budget
Pre-tax income	Production scheduled
Tax	Material order
Training expenses	Research and Development budget
Bank overdraft	Loan
Economic productivity	Term loan
Advertising prediction	
Effort sales network	

define the immediate reward as the result of the exercise in a period or step. Therefore, there is no delayed reward and, like in other classical domains like Keepaway [17], immediate rewards received in every simulation step are relevant.

III. PROPOSED ALGORITHMS FOR LEARNING VIRTUAL AGENTS

In this section we describe the new learning algorithms proposed, based on KNN and RL.

A. Adaptive KNN

In this paper, we propose a variant of KNN called Adaptive KNN (Table II). In this variant, we can distinguish two phases. In the first one, a data set C is obtained during an interaction between the agent and the environment. This data set C is composed by tuples in the form $< s, a, r >$ where $s \in S$, $a \in A$ and $r \in \Re$ is the immediate reward. In the second one, the set C obtained in the previous phase is improved during a new interaction between the agent and the environment. In each step of this second phase, the simulator returns the current state s where the agent is. The algorithm selects the K nearest neighbors to the state s in C. Among these K neighbors, it selects the tuple with the best reward obtained in the phase one. Then modify slightly the actions of this tuple and execute it. If the new reward obtained is better than the worst reward in K, it replaces the worst tuple in K with the new experience generated. Thus, the algorithm adapts the initial set C obtained in the phase one, to get increasingly better results in the second phase.

B. RL Approaches

Among many different RL algorithms, Q-learning has been widely used in the literature [20].In Q-Learning, the update function is performed following equation 1, where α is a learning parameter, and γ is a discount factor that reduces the relevance of future decisions.

$$Q(s_t, a_t) \rightarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (1)$$

Except in very small environments it is impossible to enumerate the state and action spaces. In this section we explain two new approaches for state and action space generalization problem.

TABLE II
ADAPTIVE KNN ALGORITHM

Adaptive KNN

1. Gather experience tuples
   1.1. Generate the set C of experience tuples of the type $< s, a, r >$ from an interaction of the agent in the environment, where $s \in S$, $a \in A$ and $r \in \Re$ is the immediate reward.
2. During a new interaction between the agent and the environment
   2.1 Get state s from simulator
   2.2 Select the K nearest neighbors of s in the set C
   2.2.1 For each tuple $c_i \in C$, where $c_i =< s_i, a, r >$, calculate $d(s, s_i)$
   2.2.2 Order $d(s, s_i)$ from lowest to highest
   2.2.3 Select first K tuples, C_K
   2.3. Select the tuple c_b with the best r, where $c_b =< s_b, a_b, r_b >$ and $c_b \in C_K$
   2.4. Modify a_b from c_b, $a_m = a_b \pm random\Delta$
   2.5. Execute action a_m obtaining reward $r' \in \Re$
   2.6. Update set C using the new experience
   2.6.1. Select the tuple $c_w \in C_K$ with the worst reward r_w
   2.6.2. if $r' > r_w$ then replace the tuple $c_w =< s_w, a, r_w >$ with the tuple $< s, a_m, r' >$
3. Return C

TABLE III
EXTENDED VQQL ALGORITHM

Extended VQQL

1. Gather experience tuples
   1.1. Generate the set C of experience tuples of the type $< s_1, a, s_2, r >$ from an interaction of the agent in the environment, where $s_1, s_2 \in S$, $a \in A$ and $r \in \Re$ is the immediate reward.
2. Reduce the dimension of the state space
   2.1. Let C_s the set of states in C
   2.2. Apply a feature selection approach using C_s to reduce the number of features in the state space. The resulting feature selection process is defined as a projection $\Gamma : S \rightarrow S'$
   2.3. Set $C'_s = \Gamma(C_s)$
3. Discretize the state space
   3.1. Use GLA to obtain a state space discretization, $D_{s'} = s'_1, s'_2, ..., s'_n$, $s'_i \in S'$, from C'_s.
   3.2. Let $VQ^{S'} : S' \rightarrow D_{s'}$ the function that given any state in S' returns the discretized value in D_s.
4. Discretize the action space
   4.1. Let C_a the set of actions in C
   4.2. Use GLA to obtain an action space discretization, $D_a = a_1, a_2, ..., a_m$, $a_i \in A$, from C_a
   4.3. Let $VQ^A : A \rightarrow D_a$ the function that given any state in A returns the discretized value in D_a
5. Learn the Q-Table
   5.1. Map the set C of experience tuples to a set C'. For each tuple $< s_1, a, s_2, r >$ in C, introduce in C' the tuple $< VQ^{S'}(\Gamma(s_1)), VQ^A(a), VQ^{S'}(\Gamma(s_2)), r >$
   5.2. Apply the Q-Learning update function defined in equation 1 to learn a Q table Q: $D_{s'} \times D_a \rightarrow \Re$, using the set of experience tuples C'
6. Return Q, Γ, $VQ^{S'}$, and VQ^A

1) Extended VQQL for state and action space generalization: Applying VQ techniques permits to find a more compact representation of the state and action space [6]. A vector quantizer *Q* of dimension *K* and a size *N* is a mapping from a vector (state or action) in the K-dimensional Euclidean space, R^k, into a finite set *C* containing *N* states, $Q : R^k \rightarrow C$ where $C = \{y_1, y_2, ..., y_N\}$, $y_i \in R^k$. In this way, given *C*, and a state $x \in R^k$, $VQ(x)$ assigns x to the closest state from C, $VQ(x) = arg\ min_{y \in C}\{dist(x, y)\}$.

To design the vector quantizer we use the Generalized Lloyd Algorithm (GLA). The Extended VQQL algorithm is shown in Table III.

It uses VQ to generalize the state and action spaces. In Extended VQQL algorithm, two vector quantizers are designed for each agent. The first one is used to generalize the state space and the second one is used to generalize the action space. The vector quantizers are designed from the input data C obtained during an interaction between the agent and the environment. The data set C is composed by tuples in the form $< s_1, a, s_2, r >$ where s_1 and s_2 is in the state space S, a is in the action space A and r is the immediate reward. In many problems, s is composed by a large number of features. In these cases, we suggest to apply feature selection to reduce the number of features in the state space. Feature selection is a technique of selecting a subset of relevant features for building a new subset. So feature selection is used to select the relevant features of S to obtain a subset S'. This feature selection process is defined as $\Gamma : S \rightarrow S'$. The set of states $s' \in S'$, C'_s, are used as input for the Generalized Lloyd Algorithm to obtain the first vector quantizer. The vector quantizer $VQ^{s'}$ is a mapping from a vector $s' \rightarrow S'$ into a vector $s' \in D_{s'}$, where $D_{s'}$ is the state space discretization $Ds' = s'_1, s'_2, ..., s'_n$ for $s'_i \in S'$. The set of actions $a \in A$, C_a, are used as input for the GLA to obtain the second vector quantizer.

The vector quantizer VQ^A is a mapping from a vector $a \in A$ into a vector $a \in D_a$, where D_a is the action space discretization $D_a = a_1, a_2, ..., a_m$ for $a_i \in A$. In the last part of the algorithm, the Q-table is learned from the obtained discretizations using the set C' of experience tuples. To obtain the set C' from C, each tuple in C is mapped to the new representation. Therefore, every state in C is firstly projected to the space S' and then discretized, i.e. $VQ^{S'}(\Gamma(S))$; every action $a \in A$ in C is also discretized $VQ^A(a)$.

2) CMAC-VQQL for state and action space generalization: CMAC is a form of coarse coding [20]. In CMAC the features are grouped into partitions of input state space. Each of such partition is called a *tiling* and each element of a partition is called a *tile*. Each *tile* is a binary feature. The tilings were overlaids, each offset from the others. In each tiling, the state is in one tile. The approximate value function, Q_a, is represented not as a table, but as a parameterized form with parameter vector $\vec{\theta_t}$. This means that the approximate value function Q_a depends totally on $\vec{\theta_t}$. In CMAC, each tile has associated a weight. The set of all these weights is what makes up the vector $\vec{\theta}$. The approximate value function, $Q_a(s)$ is calculated in the equation 2.

$$Q_a(s) = \vec{\theta}^T \vec{\phi} = \sum_{i=0}^{n} \theta(i)\phi(i) \quad (2)$$

The CMAC-VQQL algorithm, described in Table IV, combines two generalization techniques. It uses CMAC to generalize the state space and VQ to generalize the action space. In this case, a data set C is obtained during an interaction between the agent and the environment. This data set C is composed by tuples in the form $< s_1, a, s_2, r >$ where s_1 and s_2 is in the state space S, a is in the action space A and r is the immediate reward. In the same way that previously, s is composed by a large number of features. Feature selection is used to select a subset S' of the relevant features of S.

The set of actions $a \in A$, C_a, are used as input for the GLA to obtain the second vector quantizer. The vector quantizer VQ^A is a mapping from a vector $a \in A$ into a vector $a \in D_a$, where D_a is the action space discretization $D_a = a_1, a_2, ..., a_m$ for $a_i \in A$. Later, the CMAC is built from C'_s taking into account the obtained action space D_a. For each state variable x'_i in $s' \in S'$ the tile width and the number of tiles per tiling are selected taking into account their ranges. In our work, a separate value function for each of the discrete actions is used. In CMAC, each tile has associated a weight. The set of these weights is what makes up the vector θ. In the last part, the Q function is approximated by the equation 2.

TABLE IV
CMAC-VQQL ALGORITHM

CMAC-VQQL
1. Gather experience tuples
1.1. Generate the set C of experience tuples of the type $< s_1, a, s_2, r >$ from an interaction of the agent in the environment, where $s_1, s_2 \in S$, $a \in A$ and $r \in \Re$ is the immediate reward.
2. Reduce the dimension of the state space
2.1. Let C_s the set of states in C
2.2. Apply a feature selection approach using C_s to reduce the number of features in the state space. The resulting feature selection process is defined as a projection $\Gamma : S \rightarrow S'$
2.3. Set $C'_s = \Gamma(C_s)$
3. Discretize the action space
3.1. Let C_a the set of actions in C
3.2. Use GLA to obtain an action space discretization, $D_a = a_1, a_2, ..., a_m$, $a_i \in A$, from C_a
3.3. Let $VQ^A : A \rightarrow D_a$ the function that given any state in A returns the discretized value in D_a
4. Design CMAC
4.1. Design a CMAC function approximator from C'_s taking into account the obtained action space D_a.
5. Approximate the Q function
5.1. Map the set C of experience tuples to a set C. For each tuple $< s_1, a, s_2, r > \in C$, introduce in C' the tuple $< \Phi(\Gamma(s1)), VQ^A(C_a), \Phi(\Gamma(s_2)), r >$ where Φ is the binary vector of features
5.2. Update the vector weights θ for the action $VQ^A(C_a)$ using $\Phi(\Gamma(s_1))$, $\Phi(\Gamma(s_2))$ and r.
5.3. Apply the approximate value function defined in equation 2 to approximate the Q function for the action $VQ^A(C_a)$ using θ and $\Phi(\Gamma(C_s))$.
6. Return Q, Γ, θ, and VQ^A

IV. VIRTUAL AGENTS IN SIMBA

In the following evaluation performed, we assume that 6 companies are controlled by agents of different types. These agents are: Random Agents, that assign to each decision variable a random value following an uniform distribution; Hand-Coded Agents, that modify their decision variables by increasing their values using the Consumer Price Index (CPI); RL Agents, using the Extended VQQL and CMAC-VQQL algorithms described in Section III-B; and Adaptive KNN Agents, using the algorithm described in section III-A.

3) Executing the Extended VQQL Algorithm: Executing the Extended VQQL algorithm to learn the VQ Agents requires performing the 5 steps of the algorithm:

Step 1: Gather experience tuples. To gather experience, we perform an exploration in the domain by using hand-coded agents. Specifically, we obtain the experiences generated by a hand-coded agent managing company 1 against five hand-coded agents managing companies 2, 3, 4, 5 and 6 respectively.

Step 2: Reduce the dimension of the state space. The goal of this step is to select, from among all features in the state space, those features most related to the reward (the result of the exercise). To perform this phase, we use the data-mining tool, WEKA [22] using the attribute selection method *CfsSubsetEval*. This method evaluates the worth of a subset of attributes by considering the individual predictive ability of each feature along with the degree of redundancy between them. The resulting description of the state space after the attribute selection process is shown in Table I.

Step 3: State space discretization. Now, we use the GLA to discretize the state space.

Step 4: Discretize the action space. Again, we use GLA to discretize the action space. The action space is composed of the features shown in Table I.

Step 5: Learn the Q table. Once both the state and action spaces are discretized, the Q function is learned using the mapped experience tuples and the Q-Learning update function. The Q table is generated, composed of *n* rows (where *n* is the number of discretized states) and *m* columns (where *m* is the number of discretized actions).

4) Executing CMAC-VQQL Algorithm: Executing the CMAC-VQQL algorithm to learn the CMAC Agents requires performing the 5 steps of the algorithm as described in Table IV. Steps 1 and 2 of CMAC-VQQL are the same as steps 1 and 2 of Extended VQQL (gather experience and the reduction of the dimension of the state space). Step 3 of CMAC-VQQL (action space discretization) is also the same as step 4 of Extended-VQQL. Step 4 is the design of the CMAC function approximator. In our experiments we use single-dimensional tilings. For each state variable, 32 tilings were overlaid, each offset from the others by 1/32 of the tile width. For each state variable, we specified the width of the tiles based on the width of the generalization that we desired. In the experiments we use three different configurations. The size of the primary vector θ in Configuration #1 is 754272 ($x_{1_{tiles}} + x_{2_{tiles}} + + x_{12_{tiles}}$), in Configuration #2 is 1364320, in Configuration #3 is 2440704. In our work, we use a separate value function for each of the generalized actions. Last, step 5 of the algorithm, learning the Q approximations, can be performed.

A. Adaptive KNN in SIMBA

To apply the Adaptive KNN algorithm to create a *SIMBA* software agent, we use the same state space, action space, and transition and reward functions that for the RL agent. We also use the same experience tuples than for the RL agent, although in the learning process, the set is updated following step 6 of the algorithm (as described in Table II).

TABLE V
RESULTS FOR DIFFERENT CONFIGURATIONS OF EXTENDED VQQL (IN MILLIONS OF EUROS).

Decisions / States	128		64		32	
	Mean	Std	Mean	Std	Mean	Std
128	4,3	1,73	5,43	4,2E-04	7,73	0,09
64	6,21	3,15	7,51	0,32	**8,14**	0,51
32	4,94	3,68	5,69	0,06	7,62	0,28

TABLE VI
RESULTS FOR DIFFERENT CONFIGURATIONS OF CMAC-VQQL (IN MILLIONS OF EUROS).

Decisions / Configuration	64		32		8	
	Mean	Std	Mean	Std	Mean	Std
1	4,87	1,62	6,49	0,04	**7,0**	0,12
2	6,23	0,13	5,82	0,90	6,25	0,37
3	5,26	0,20	5,95	3,2E-04	6,24	0,96

V. RESULTS

In the experiments, the learning agent always manages the first company of the six involved in the simulations. Each experiment consists of 10 simulations or episodes with 20 rounds and we obtain the mean value and the standard deviation for the result of the exercise during the 20 periods. In this situation, a hand-coded agent that manages the company 1 against five hand-coded agents that manage companies 2, 3, 4, 5 and 6 respectively obtains a mean value of the result of the exercise of 2,901,002.13 euros. A random agent in the same situation obtains -2,787,382.78 euros.

In the experiments with human experts, simulations have 8 rounds.

A. RL and KNN Results

In the first set of experiments we use the Extended VQQL algorithm to learn an agent that manages company 1 and plays against five hand-coded agents that manage companies 2, 3, 4, 5 and 6 respectively. The results for different discretizations size of the state (rows) and action (columns) spaces are shown in Table V.

The best result is obtained when we use a vector quantizer of 64 centroids (or states) to generalize the state space and a vector quantizer of 32 centroids (or actions) to generalize the action space.

In the second set of experiments we use the CMAC-VQQL algorithm. The results for the different CMAC configurations described in section IV-4 (rows) combined with the different sizes of the action space obtained by VQ (columns) are shown in Table VI.

The best result is obtained when we use the Configuration #1 of CMAC to generalize the state space and a vector quantizer of 8 centroids to generalize the action space. This value is smaller than the obtained with Extended VQQL but, again, all the configurations obtain better results than the hand-coded agent.

In the next set of experiments we use the KNN algorithm to build an agent. The results for the different KNN configurations are shown in Table VII.

TABLE VII
RESULTS FOR DIFFERENT CONFIGURATIONS OF KNN (IN MILLIONS OF EUROS).

K	5		10		15	
Learning	Mean	Std	Mean	Std	Mean	Std
Adaptive	6,44	3,99	**9,81**	0,21	**9,89**	0,32
No adaptive	7,86	1,15	5,20	1,11	7,47	4,36

The columns of Table VII show different results for different values of K (5, 10 and 15 respectively). The first row presents the results of the Adaptive KNN algorithm, as it was described in Table II. The second row shows the results of a classical KNN approach, without the adaptation of the training set, i.e. without executing the steps five and six of the Adaptive KNN algorithm. The best results are obtained with the adaptive version, for K=10 and K=15. In these cases, we obtain a mean value for the result of the exercise of 9,8 millions of euros, which is higher than the ones obtained with RL.

In previous experiments, the learning agent always learned to manage the first company of the six involved in the simulations. However, the behavior of each company depends on their initial states and of historical data (periods -1, -2, etc). Therefore, learning performance may vary from one company to other. To evaluate this issue, we repeat the learning process for the best learning configurations, for each of the six companies. Each experiment consists of 10 simulations with 20 rounds and we obtain the mean value and the standard deviation for the result of the exercise during the 20 periods. The results shown in Figure 2 demonstrate that the Extended VQQL agent and Adaptative KNN agent obtain similar results, and both obtain better results than the hand-coded agent.

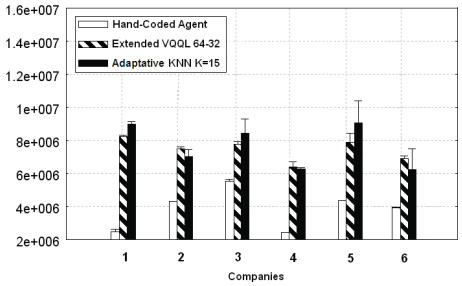

Fig. 2. Mean value and Standard deviation for the result of the exercise.

Now, we compare the behavior of the best RL agent with the behavior of the best Adaptative KNN agent obtained in previous experiments. In this experiment, all the companies have the same initial state and historical data, so the result is independent of the company managed. This experiment consists of 10 simulations with 20 rounds and we obtain the mean value and the standard deviation for the result of the exercise during the 20 periods. Figure 3 shows the mean value and the standard deviation for each kind of agent.

For the Adaptive KNN agent, the average value grows from the first period, and raises up to 16 millions of euros. However,

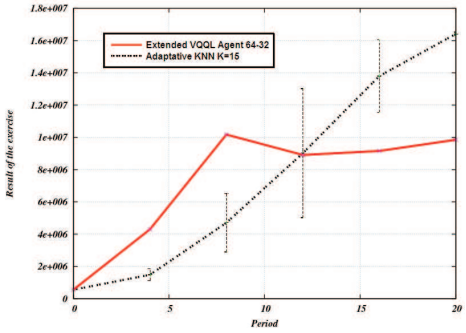

Fig. 3. RL Agents vs. Adaptive KNN Agents

TABLE VIII
RESULTS FOR INCREMENTAL DECISION STRATEGY (IN MILLIONS OF EUROS)

Simulation / Agent	10%	20%
Extended VQQL 64-32	7,27	7,28
Adaptive KNN K=15	1,58	1,58
Human Expert	0,56	-0,18

we see that standard deviation is very high, so the behavior of the agent managing different companies is very different. The result for the Extended VQQL agent have two behaviours well differentiated: before period 8, and after period 8. In the first part, the result of the exercise always grows up, and dominates the result of the Adaptive KNN agent. However, from period 8, the result of the exercise for the Extended VQQL agent stabilizes to a value of around 10 millions, and it is dominated by the other agent from period 10. Interestingly, we have revised all the simulations performed, and this behavior always appears. We believe that the RL agent is affected by the CPI and the evolution of the market and, with time, the actions obtained by the VQ algorithm becomes old-fashioned (note that 8 periods are equivalent to two years). Therefore, if we focus in the early periods, typically the RL agents behave better than the KNN ones.

B. RL and KNN Agents vs. Human Experts

In this section, we present experiments where software agents play against a human expert during 8 periods. The human expert actually is an associate full time professor in Strategic and Business Organization at Universidad Autónoma de Madrid (UAM), where he is Director of Master of Business Administration (Executive) and Director of Doctorate Program of Financial Economics.

In all the experiments, we use the best RL and Adaptive KNN agents obtained in the previous section. In the first experiment, the human expert uses the incremental decision strategy, described in section II. The results are shown in table VIII.

In this case, the Extended VQQL agent obtains the best results. Furthermore, given that only 8 episodes are run, the RL agent performs much better than the Adaptive KNN agent.

TABLE IX
AGENTS VS. HUMAN EXPERT (IN MILLIONS OF EUROS)

Agent \ Simulation	1	2
Extended VQQL 64-32	5,36	6,09
Adaptive KNN K=15	3,47	2,53
Human Expert	-0,32	-1,30

The human expert obtains the worst results (independently of the increment used).

In the second experiment, two different simulations with 8 rounds each are performed. The human expert combines the use of the different business strategies described in section II. The results are shown in table IX.

In all the experiments, the software agents obtain better results than the human expert. From a qualitative point of view, the virtual agents usually compete in the same market scope. They are very effective and efficient, been almost impossible to beat them under the parameter setting used in these simulations. The best strategies usually make decisions in different market scopes, using high or low strategies (for instance, low cost or differentiation and specialization). It means that using more competitive strategies, the gap between the performance of the virtual agents and the human experts could be reduced.

VI. BACKGROUND

Business gaming usage has grown globally and has a long and varied history [4]. The first modern business simulation game can be dated back to 1932 in Europe and 1955 in North America. In 1932, Mary Birshstein, while teaching at the Leningrad Institute, got the idea to adapt the concept of war games to the business environment. In North America the first business simulator dates back to 1955, when RAND Corporation developed a simulation exercise that focused on the U.S. Air Force logistics system [9]. However, the first known use of a business simulator for pedagogical purposes in an university course, was at the University of Washington in a business policy course in 1957 [21].

From this point, the number of business simulation games in use grew rapidly. A 2004 e-mail survey of university business school professors in North America reported that 30.6% of 1,085 survey respondents were current business simulation users, while another 17.1% of the respondents were former business game users [5].

Over the years Artificial Intelligence (AI) and simulation have grown closer together. AI is used increasingly in complex simulation, and simulation is contributing to the development of AI [24]. The need for increased level of reality and fidelity in domain-specific games calls for the use of methods that bring realism and intelligence to actors and scenarios (also in business simulators). Intelligent software agents, called "autonomous" avatars or virtual players, are now being embodied in business games. Software agents can interact with each other and their environment producing new states, business information and events. In addition, these agents not only provide information but also may affect the environment and direction of the simulation [19].

Machine learning techniques (decision trees, reinforcement learning,...) have been used widely to develop software agents [18]. In [19] for example, the software agents uses decision trees to learn different behaviors. In this case, virtual players could take on the role of an executive or salesperson from a supplier firm, a union leader, or any other role relevant to the simulation exercise. In [10] the learning agents uses a typical genetic-based learning classifier system, XCS (*eXtended learning Classifier System*). In that work, RL techniques are used, allowing decision-making agents to learn from the reward obtained from executed actions and, in this way, to find an optimal behavior policy. In stochastic business games, the players take actions in order to maximize their benefits. While the game evolves, the players learn more about the best strategy to follow. With this, RL can be used to improve the behavior of the players in a stochastic business game [13]. However, in all these cases, the business simulator games used did not involve the huge state and action spaces that SIMBA involves.

In complex domains with large state and action spaces is necessary to apply generalization techniques such as VQ or CMAC. VQ has been used successfully in many other domains [6]. In addition, CMAC [17] are extensively used to generalize the state space, but the research on problems where the actions are chosen from a continuous space is much more limited. KNN has also been used in the scope of Business Intelligence. In [16], the authors investigates the relationship among corporate strategies, environmental forces, and the Balanced Scorecard (BSC) performance measures using KNN. In this case, the authors used all time the same initial set of experience and they did not try to adapt it, using the new experience generated during the game.

An important issue that make SIMBA different from other classical RL domains (like Keepaway [17]) is that it is not defined a priori as a cooperative or competitive domain. In SIMBA, the number of adversaries is very high, and the number of variables involved in the state and action space, too. In addition, in SIMBA software agents can play against humans. It is hard to find all these issues in other classical domains. So the learning process in SIMBA represent a real challenge.

VII. CONCLUSION

This paper introduces SIMBA as a business simulator which architecture enables different players, including both software agents and human players, to manage companies in different markets. The simulator generates a competitive environment, where the different agents try to maximize their companies' profits. SIMBA represents a complex domain with large state and action spaces. Therefore, the learning approaches applied to generate the virtual agents must handle that handicap. We have demonstrated that the proposals presented, based on Lazy Learning and RL, achieve the goal of being very competitive

when compared with previous hand-coded strategies. Furthermore, we demonstrate that when competing with a human expert, which follows classical management strategies, the learning agents are able to outperform the behavior of the human.

In the case of RL, the choice of the generalization method have a strong effect on the results that we obtain. For this reason, the state and action space representation is chosen with great care, and we have proposed two new methods: Extended-VQQL and CMAC-VQQL. This is the first time that VQ is used to discretize the action space, and some preliminary results have shown that it is also useful in other domains, like autonomous helicopter control. The challenging results obtained by the learning approaches to generate virtual agents in SIMBA offers promising results for Autonomous Decision Making.

ACKNOWLEDGMENT

This work has been partially supported by the Spanish MICINN project TIN2008-06701-C03-03 and by the Spanish TRACE project TRA2009-0080. The authors would like to thank the people from Simuladores Empresariales S.L.

REFERENCES

[1] J. S. Albus. A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Journal of Dynamic Systems, Measurement, and Control*, 97(3):220–227, 1975.

[2] F. Borrajo, Y. Bueno, I. de Pablo, B. n. Santos, F. Fernández, J. García, and I. Sagredo. Simba: A simulator for business education and research. *Decision Support Systems*, June 2009.

[3] M. Buro. Improving heuristic mini-max search by supervised learning. *Artificial Intelligence*, 134:85–99, 2002.

[4] A. J. Faria, D. Hutchinson, W. J. Wellington, and S. Gold. Developments in business gaming: A review of the past 40 years. *Simulation Gaming*, 40(4):464–487, August 2009.

[5] A. J. Faria and W. J. Wellington. A survey of simulation game users, former-users, and never-users. *Simul. Gaming*, 35(2):178–207, 2004.

[6] F. Fernández and D. Borrajo. Two steps reinforcement learning. *International Journal of Intelligent Systems*, 23(2):213–245, 2008.

[7] F.-H. Hsu. *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Princeton University Press, Princeton, NJ, USA, 2002.

[8] R. Hunicke and V. Chapman. Ai for dynamic difficulty adjustment in games. 2004.

[9] J. R. Jackson. Learning from experience in business decision games. *California Management Review*, 1:23–29, 1959.

[10] M. Kobayashi and T. Terano. Learning agents in a business simulator. In *Proceedings 2003*. IEEE International Symposium on Computational Intelligence in Robotics and Automation, 2003.

[11] R. E. Miles and C. C. Snow. *Organizational strategy, structure, and process*. McGraw-Hill, New York, 1978.

[12] M. E. Porter. *Competitive Advantage: Creating and Sustaining Superior Performance*. Free Press, New York, 1 edition, June 1985.

[13] K. K. Ravulapati and J. Rao. A reinforcement learning approach to stochastic business games. *IIE Transactions*, 36:373–385, 2004.

[14] J. Santamaria, R. Sutton, and A. Ram. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior*, 6:163–217, 1998.

[15] J. Schaeffer and H. J. van den Herik. Games, computers, and artificial intelligence. *Artif. Intell.*, 134(1-2):1–7, 2002.

[16] M. H. Sohn, T. You, S.-L. Lee, and H. Lee. Corporate strategies, environmental forces, and performance measures: a weighting decision support system using the k-nearest neighbor technique. *Expert Syst. Appl.*, 25(3):279–292, 2003.

[17] P. Stone, R. S. Sutton, and G. Kuhlmann. Reinforcement learning for RoboCup-soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[18] P. Stone and M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, June 2000.

[19] G. J. Summers. Today's business simulation industry. *Simul. Gaming*, 35(2):208–241, 2004.

[20] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. Mit Pr, May 1998.

[21] H. J. Watson. *Computer Simulation in Business*. John Wiley & Sons, Inc., New York, NY, USA, 1981.

[22] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, second edition, June 2005.

[23] J. Yang, S. Min, C.-O. Wong, J. Kim, and K. Jung. Dynamic game level generation using on-line learning. In *Edutainment'07: Proceedings of the 2nd international conference on Technologies for e-learning and digital entertainment*, pages 916–924, Berlin, Heidelberg, 2007. Springer-Verlag.

[24] L. Yilmaz, T. Ören, and N.-G. Aghaee. Intelligent agents, simulation, and gaming. *Simul. Gaming*, 37(3):339–349, 2006.