Evolution of Reactive Rules in Multi Player Computer Games Based on Imitation

Steffen Priesterjahn[1], Oliver Kramer[2], Alexander Weimer[1], and Andreas Goebels[2]

[1] Department of Computer Science
[2] International Graduate School on Dynamic Intelligent Systems
University of Paderborn, 33098 Paderborn, Germany, swarmgroup@upb.de

Abstract. Observing purely reactive situations in modern computer games, one can see that in many cases few, simple rules are sufficient to perform well in the game. In spite of this, the programming of an artificial opponent is still a hard and time consuming task in the way it is done for the most games today. In this paper we propose a system in which no direct programming of the behaviour of the opponents is necessary. Instead, rules are gained by observing human players and then evaluated and optimised by an evolutionary algorithm to optimise the behaviour. We will show that only little learning effort is required to be competitive in reactive situations. In the course of our experiments our system proved to generate better artificial players than the original ones supplied with the game.

1 Introduction

Modern computer games have become much more complex and dynamic than their ancestors. With this increase of complexity, the demands for the design of artificial game players have also become higher and higher. On the other hand A.I. routines are not allowed to consume much computing time, since most of this is reserved for the graphics and the user interface. So, programming A.I. routines for modern computer games is very hard, time consuming, and therefore expensive.

We think that much of this programming is unnecessary, since many situations in computer games (e.g. combat) can be handled very well by reactive rules. This is especially the case in so called "first person shooter" (FPS) games in which the player is running through three-dimensional environments (maps) and has to defend himself against other artificial or human players.

In this paper, we propose a system for learning such reactive rules for combat situations by observing the behaviour of human players. Based on this data evolutionary algorithms are used to select the best and most important rules and to optimise the behaviour of the artificial player. The modelling of these rules is presented in section 3. Section 4 gives information about the evolutionary algorithm which was used to optimise the rule sets. Finally, section 5 presents the results we gained from our experiments and in which our agents eventually were able to defeat the artificial players provided by the game.

2 Related Work

Using computer games for artificial intelligence research has become more common in recent years. Games can be used very well as a testbed for different A.I. techniques. Especially three-dimensional action games are frequently used in current research.

An interesting approach for a learning agent has been proposed by Laird et al. in [6]. Their agents try to anticipate the actions of the other players by evaluating what they would do, if they were in the position of the other player. In a later version reinforcement learning was added to their approach [9]. Hawes [4] uses planning techniques for an agent. It uses times of low activity to plan extensive behaviours and generates only short plans if no time is available. Nareyek [7, 8] has also implemented an agent which uses planning and local search. Another interesting approach has been applied by Norling [10], in which a BDI-model (Belief-Desire-Intention) is used to model a human-like agent.

The research of Thurau et al. [13–15] has much in common with the approach presented in this paper, because it also relies on imitation. [13] emphasises the imitational aspect of their approach. However, the used representation and learning techniques are different. In [14] and [15], neural nets which are trained on data gained from human players and neural gas to represent the structure of the environment are used to learn gaming behaviour. Our approach is oriented to evolution strategies as described in [2] and [12].

Since this paper is focussed on the training of single agents, we will just give some short examples on the research of teams of game agents. Bakkes et al. [1] use evolutionary algorithms to evolve team strategies for the "capture the flag" game. In [5], Kaminka et al. use arbitration to negotiate team strategies. Priesterjahn et al. [11] have developed a system which enables the agents to avoid dangerous areas in the environment based on the past experience of the team. The concept of stigmergy[3] is used to accomplish such behaviour.

3 Basic Modelling

In this section the modelling of the rules and the environment of the player is described. Since it is common to call artificial players in a computer game *bots* or *agents*, we will also use these denotations. We have chosen the game Quake3 (see figure 1) as the basis of our research, because it allows fundamental modifications to the game. Quake3 is a first person shooter game and is focussed on multi player gaming. This means that several human players are competing in a highly dynamic environment. Therefore, a bot has to be able to compete with the human players, because its purpose is to substitute one of them. For our first experiments, we have modified the game so that jumping or ducking is not allowed. We also use a map which does not have several ground levels. So, looking up and down is not needed.

[3] Stigmergy denotes that information is not exchanged directly but through the environment.

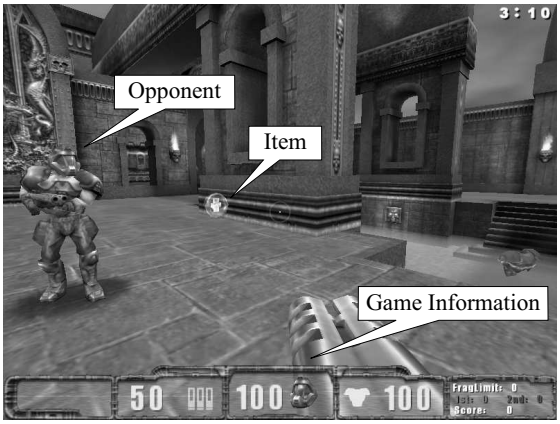

Fig. 1. A scene from Quake3

3.1 The Rules

We use reactive rules to describe the agent behaviour. Each rule is defined by an input and an adequate output. For the input, we compress the incoming information, to reduce the search space. The input does not need to hold every possible information. For example, only the vicinity of the bot is important. Furthermore, though Quake3 runs in a three-dimensional environment, the game is mostly played in two dimensions, because the players can only walk on the ground. Therefore, we chose to use a two-dimensional representation for the input in the form of a matrix. This matrix is constructed as follows.

The environment of an agent is segmented into a grid of quadratic regions lying on the floor (see figure 2). Every grid field corresponds to a position in the matrix. The bot is positioned at the centre of the grid. The alignment of the grid is always relative to the bot. So, if the bot moves, the grid will be rotated and shifted to fit these movements. Every grid field is always placed at the same relative position to the agent. The size of the grid is limited to the size of the matrix. So, it only covers the vicinity of a bot.

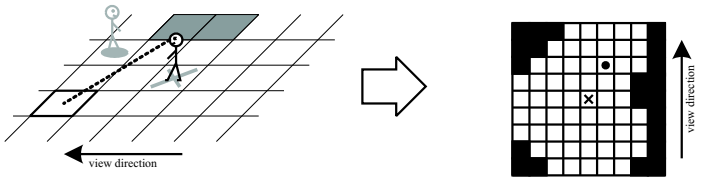

Fig. 2. Obtaining the grid from the player. (x - player, • - opponent)

In each acting frame of the agent, it "traces" to the center of every grid field on the floor of the environment. This can be compared to using a laser sensor. In each trace a ray is sent from the head of the bot. If this ray reaches the center

of the grid field it was sent to, the corresponding value of the matrix is set to a value which indicates that this field is empty. Otherwise, the field is indicated as filled. If a field is occupied by an opponent, a corresponding value will be written into the matrix. The central field is always regarded as empty and can not be changed. In the following, we will also use the term *grid* for this matrix. A detailed and formal description of this grid is given in the following definition.

Definition 1 (Grid)
A grid G is a matrix $G = (g_{i,j})_{1 \leq i,j \leq n} \in \mathbb{N}_0^{n \times n}$, $n \in \mathbb{N}$ with $n \equiv 1 \bmod 2$ and

$$g_{i,j} = \begin{cases} 0, & \textit{if the field is occupied} \\ 1, & \textit{if the field is empty} \\ 50, & \textit{if the field contains an opponent.}^4 \end{cases}$$

$\mathcal{G}$ denotes the set of all grids.

The output of a rule represents a *command* which is executed by the bot and is defined as follows.

Definition 2 (Command, Rule)
A command C is a 4-tuple $C = (f, r, \varphi, a)$ with $f, r \in \{-1, 0, 1\}$, $a \in \{0, 1\}$ and $\varphi \in [-180°, 180°]$. The interpretation of these variables is as follows.

$$f = \begin{cases} 1, & \textit{move forward} \\ 0, & \textit{no movement} \\ -1, & \textit{move backward} \end{cases} \qquad r = \begin{cases} 1, & \textit{move to the right} \\ 0, & \textit{no movement} \\ -1, & \textit{move to the left} \end{cases}$$

$$a = \begin{cases} 0, & \textit{do not attack} \\ 1, & \textit{attack} \end{cases} \qquad \varphi = \textit{alteration of the yaw angle}$$

$\mathcal{C}$ denotes the set of all Commands.

A rule $R : \mathcal{G} \to \mathcal{C}$ maps a grid to a command. $\mathcal{R}$ denotes the set of all rules.

3.2 Creating the Rule Base

Though we already have reduced the complexity of the problem, the search space is still very large. [5]. Therefore, we chose to generate a basic rule set by recording human players. This is simply done by letting them play against each other and by recording their grid-to-command matches for every frame of the game. So, rules which are executed very often, are put more often into our rule base.

In the first step, certain behaviours of the players will then be imitated by our bots. Then the selection of the appropriate rules from the rule base and

[4] A value of 50 has been chosen to emphasise the position of an opponent.
[5] at least $3^{n \cdot n}$ for a $n \times n$ grid

the performance of the agents is optimised by an evolutionary algorithm. This approach has the advantage that certain behaviours can be presented to the bot, from which it learns to use the best in relation to its fitness function. In this way an agent can be trained to have a certain behaviour without programming it manually.

3.3 Rule Comparison

Each bot owns an individual set of rules from which it chooses the best one for its current situation. For this selection process, it has to be able to compare different grids to find the rule with the grid which fits best to its current input. Therefore, a distance measure between two grids has to be introduced. We chose to use the euclidean distance.

Definition 3 (Euclidean Distance)
The euclidean distance between two $n \times n$-grids G and G' is defined by

$$\mathrm{dist}(G, G') = \sum_{1 \leq i,j \leq n} (g_{i,j} - g'_{i,j})^2.$$

However, this distance does not take into account the rough similarity of two grids. For example the following matrices A, B and C have the same euclidean distance, though A and B are more similar to each other.

A	B	C	A'	B'	C'
1 0 0	0 1 0	0 0 0	1 0.5 0	0.5 1 0.5	0 0 0
0 0 0	0 0 0	0 0 0	0.5 0.2 0	0.2 0.5 0.2	0 0.2 0.5
0 0 0	0 0 0	0 0 1	0 0 0	0 0 0	0 0.5 1

If A, B and C are smoothed with a smoothing operator, matrices like A', B' and C' will result. Now the euclidean distance between A' and B' is smaller as between A' and C' or B' and C'. Therefore, we convolve the grid with the Gaussian smoothing filter. This operator is commonly used in image processing and its filter matrix $F = (f_{i,j})_{-r \leq i,j \leq r}$ is defined as

$$f_{i,j} = e^{-\frac{i^2+j^2}{r^2}},$$

where r is the radius of the filter. For further information about convolution, see [3]. We define the *euclidean gauss distance* as follows.

Definition 4 (Gaussian Grid, Euclidean Gauss Distance)
Let G be a $n \times n$-grid. Then, $G_g = (g^g_{i,j})_{1 \leq i,j \leq n} \in \mathbb{R}_{\geq 0}^{n \times n}$ denotes the result of a convolution of G with a Gaussian filter of radius $r \in \mathbb{R}_{\geq 0}$. The euclidean gauss distance dist_g *between two grids G and G' is defined as*

$$\mathrm{dist}_g(G, G') = \mathrm{dist}(G_g, G'_g).$$

We used a radius of $r = 5$ for our Gaussian filter, which results in a 5×5 filter matrix.

4 Rule Evolution

During the optimisation phase of our approach the rule base is optimised with an evolutionary algorithm (EA). Every bot has a rule set $\{R_1, ..., R_k\} \in \mathcal{R}^k$ with a fixed size $k \in \mathbb{N}$. At the beginning the first individuals are initialised with randomly chosen rules from the rule base, which is created as described in section 3.2. Then crossover and mutation are used to select the best rules and to gain further optimisation of the performance of the bots.

Population and Structure:
Concerning the population structure and the selection scheme of our evolutionary algorithm we use a (μ, λ)-EA oriented to evolution strategies. The size of the parental population is $\mu \in \mathbb{N}$. In each generation $\lambda \in \mathbb{N}$ offspring individuals are produced applying the variation operators crossover and mutation.

Crossover:
For the crossover, two parents are chosen randomly with uniform distribution from the parental population. Let $\{R_1, ..., R_k\} \in \mathcal{R}^k$ and $\{R'_1, ..., R'_k\} \in \mathcal{R}^k$ be the rule sets of the parents. Then, for the rule set of the offspring $\{O_1, ..., O_k\} \in \mathcal{R}^k$, rule O_i is randomly chosen from $\{R_i, R'_i\}$ with uniform distribution. So, crossover effects the structure of the rule sets.

Mutation:
In contrast to crossover, the mutation operator effects the structure of the rules itself. All changes are made with the same probability p_m and uniform distribution. For the grid, a grid field can be changed from empty to full or vice versa. The position of an opponent on the grid can be changed to one of the neighbouring grid fields, though it can not be moved beyond the grid borders. For the command (f, r, a, φ) of a rule, f, r and a can be set to another value. The alteration of the view angle φ can be changed by adding a random angle $\Delta\varphi \in [-5°, 5°]$

Simulation and Fitness Calculation:
The fitness of each bot is evaluated by letting it play and apply its rule set for a simulation period of n_{sim} seconds. The summed health loss of the opponents $h_{\text{opp}} \in \mathbb{N}_0$ and the health loss of the bot $h_{\text{own}} \in \mathbb{N}_0$ is counted and integrated into the fitness function

$$f = w_{\text{opp}} h_{\text{opp}} - w_{\text{own}} h_{\text{own}}.$$

Health loss of the opponent increases, own health loss decreases the fitness of the agent.

Selection:
We use the plus-selection scheme. The parental population $P_p(t+1)$ for generation $t+1 \in \mathbb{N}$ consists of the μ best individuals of the current offspring $P_o(t)$

and the last parental population $P_p(t)$. In this selection scheme parents with superior fitness values can survive as long as their fitness belongs to the μ best ones.

5 Results

A series of experiments showed that our agents succeed in imitating the behaviour of the base players and that they are able to improve their behaviour beyond their basis. The selection of the following parameters is a result of our initial experiments.

5.1 Experimental Setup

We used an evaluation timespan of 45 seconds for each individual, which was just long enough to provide reliable results. For a population of 40 individuals, this results in an evaluation timespan of 30 minutes for each generation.

To have a constant opponent for our experiments, we used an original Quake3-bot for the training. We also used the behaviour of the Quake3-bot to generate the rule base. This was accomplished by letting it play against itself. We also chose the Quake3-bot as the basis, because we wanted to find out if our bot would be able to surpass its basis and to generate new behaviour. We made experiments with a dense grid of 25×25 fields and a coarse grid of 15×15 fields, which were both sized to represent approximately 15×15 metres in Quake3.

The size of the rule set of an agent was set to 50 rules and 400 rules in different experiments, to study the influence of this parameter on the behaviour of the agent. On the one hand it was interesting to see how much rules are necessary to gain competitive behaviour. On the other hand we wanted to find out if more complex behaviour can be gained by using large rule sets.

As it was already indicated above we chose a population of 40 individuals for each generation, consisting of $\mu = 10$ parents and $\lambda = 30$ offspring individuals. We used these values because they are small enough to allow a relatively fast evolution and big enough to retain diversity.

The last parameter we had to choose was the mutation probability p_m. In early experiments we found out that using a mutation probability that is too high can destroy good behaviours that have already been learned. So, we decided to use a relatively small mutation probability of $p_m = 0.005$. This results in an algorithm that is mainly focussed on selecting the best rules and not on learning new behaviour. However, we also conducted experiments with $p_m = 0.1$. To avoid the destruction of important rules in these experiments, we chose to apply the mutation only to the command part of a rule and not to the grid. Thus, the rule inputs remained untouched. This approach implies the assumption that all important states are already included into the rule base.

We decided to stop the evolution after a fixed number of steps, which was based on our initial experiments. Table 1 gives an overview of the conducted experiments.

Table 1. Experimental setup

exp #	grid density	rule set	grid mutation	p_m
1	25×25	50	yes	0.005
2	25×25	400	yes	0.005
3	15×15	50	yes	0.005
4	15×15	400	yes	0.005
5	25×25	50	no	0.1
6	25×25	400	no	0.1

5.2 Equally Weighted Fitness

In our first series of experiments we calculated the fitness by $f = 1 \cdot h_{\text{opp}} - 1 \cdot h_{\text{own}}$. If our bot has caused more damage to its opponent than it has received from it, the fitness will be greater than zero.

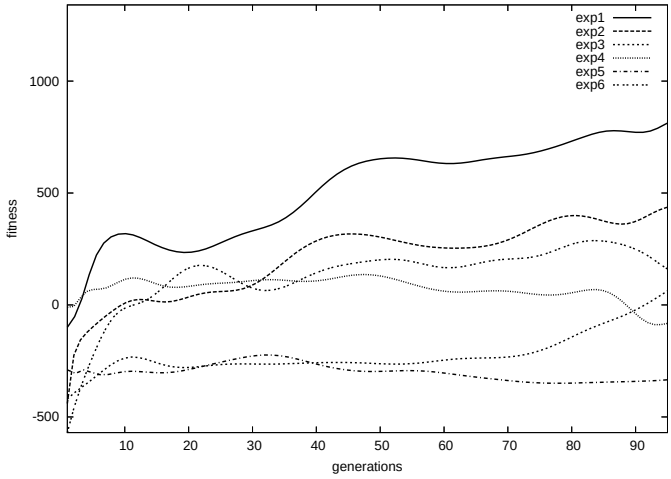

Fig. 3. Smoothed plots of the fitness progression ($w_{\text{opp}} = w_{\text{own}} = 1$)

We obtained several interesting results. First of all, our agents performed well and the best of them were able to defeat the Quake3-bot already after 5 generations. After 20 to 30 generations the bots reached satisfiable and competitive behaviour in almost all experiments. Figure 3 shows the fitness progression of the best fitness of each generation for these experiments. We smoothed the plots with a Bezier curve to improve readability.

Figure 4 shows the fitness progression of the best experiment (exp1). The best, worst and average fitness of each generation is displayed. These plots are again smoothed with a Bezier curve. Furthermore, the real data for the best fitness of each generation is displayed for comparison. It should be noted that even the average fitness grew above zero in this experiment.

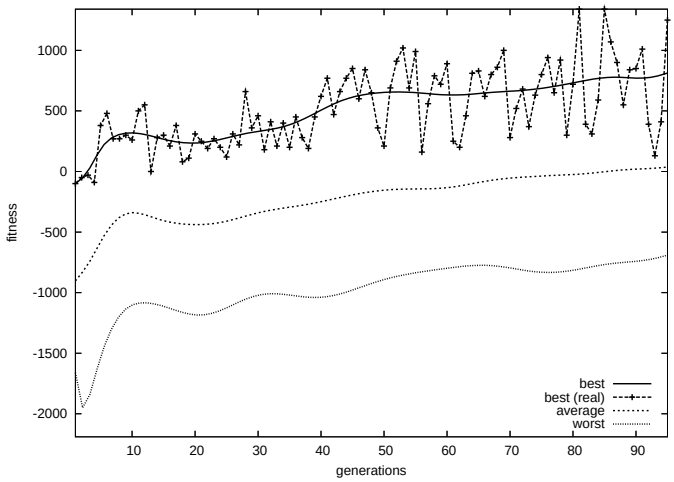

Fig. 4. Performance of the best experiment ($w_{\text{opp}} = w_{\text{own}} = 1$)

At the beginning the agents imitated some behaviours of the Quake3-bot closely. However, there were individuals which ran into corners and could not get out of them. These ones were sorted out by the EA after few generations. In the course of the evolution the agents even developed new behaviour. For example the best ones began to take cover behind a column. This behaviour is not implemented in the Quake3-bot. In the experiments with the higher mutation probability the bots took more freedom in their movements. However, their performance never got as good as the performance of the agents with grid mutation. The usage of a more dense grid resulted in an improvement of the performance.

Concerning the number of used rules, we made further experiments with a trained agent and a rule set of 400 rules by letting it play for one hour. It used more than 300 of its rules. However, experiments 1 and 2 show, that limiting the number of rules to 50 can be advantageous. Observing the behaviour of the agents, the ones with 50 rules also behaved more efficiently and flawlessly. An exception is experiment 5 in which the algorithm got stuck in a local maximum. In this setting the agents learned to run away from their opponent, thus applying only little damage to it.

5.3 Aggressive Fitness

To investigate the influence of the fitness function on the behaviour of the agents we conducted a second series of experiments. We chose $f = 2 \cdot h_{\text{opp}} - 1 \cdot h_{\text{own}}$ as the fitness function, to suppress the tendency to run away. So, neither the fitness values of these experiments can be directly compared to the ones obtained above, nor a comparison with the Quake3-bot based on the fitness values is possible in a direct way. Another series of experiments with the same parameter sets as above was conducted.

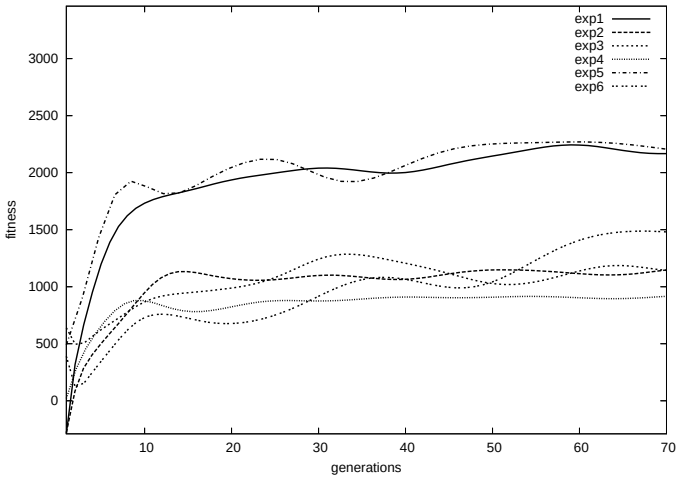

Fig. 5. Smoothed plots of the fitness progression ($w_{\text{opp}} = 2, w_{\text{own}} = 1$)

Figure 5 again shows the smoothed progression of the best fitness of each generation. Again experiment 1 provided good results. However, this time experiment 5 also delivers good results. The experiments without grid mutation showed no disadvantage in this setup. Concerning the rule set size, the agents using the smaller sets performed better than the ones using a large set. Again, the coarse grid produced inferior results. There were some agents which were able to defeat the Quake3-bots in all experiments. Though, after longer[6] evolution the bots tended to act very aggressively and to disregard their own health. So, the fitness function can be used to learn certain aspects of the behaviour. The overall convergence in these experiments was better than in the equally weighted setting.

6 Conclusion & Future Work

In the experiments, our agents were able to behave in the same way as the original players already after few generations. They also were able to improve their performance beyond their basis and to develop new behaviour. We have presented a system which uses evolutionary algorithms to learn rules for successful reactive behaviour in multi player games based on imitation. This approach can be used to train certain aspects of the behaviour of an artificial opponent based on the imitation of other players and to emphasise desired behaviours. Our approach has also turned out to prevent disadvantageous behaviour, because such behaviour, e.g. getting stuck in corners or standing still, has been eliminated in all experiments after at most 20 to 30 generations.

In the future we will conduct further experiments to find out more about the parameter dependency of our approach and to get statistically more significant

[6] about 50 generations

data. Other representations will also be studied. We will also apply preprocessing steps to our rule base, like data mining and clustering techniques, to find out important states in our data and to improve the imitation. Furthermore, we want to use reinforcement learning for a more complex, preprocessed representation to learn more complex behaviour.

References

1. S. Bakkes, P. Spronck, and E. Postma. TEAM: The Team-Oriented Evolutionary Adaptability Mechanism. In *Proceedings of the ICEC*, pages 273–282, 2004.
2. H.-G. Beyer and H.-P. Schwefel. Evolution strategies – A comprehensive introduction. *Natural Computing*, 1:3–52, 2002.
3. R. C. Gonzalez and P. A. Wintz. *Digital Image Processing.* Addison Wesley, 1992.
4. N. Hawes. An Anytime Planning Agent For Computer Game Worlds. In *Proceedings of the Workshop on Agents in Computer Games at The 3rd International Conference on Computers and Games*, pages 1–14, 2002.
5. G. Kaminka, J. Go, and T. Vu. Context-dependent joint-decision arbitration for computer games, 2002.
6. J. Laird. It Knows What You're Going to Do: Adding Anticipation to a Quakebot. In *AAAI 2000 Spring Symposium Series: Artificial Intelligence and Interactive Entertainment: AAAI Technical Report SS-00-02*, 2000.
7. A. Nareyek. A Planning Model for Agents in Dynamic and Uncertain Real-Time Environments. In *Proceedings of the Workshop on Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments at the Fourth International Conference on Artificial Intelligence Planning Systems*, pages 7–14. AAAI Press, 1998.
8. A. Nareyek. Constraint-Based Agents - An Architecture for Constraint-Based Modeling and Local-Search-Based Reasoning for Planning and Scheduling in Open and Dynamic Worlds. In *Künstliche Intelligenz 2*, pages 51–53, 2002.
9. S. Nason and J. Laird. Soar-RL: Integrating Reinforcement Learning with Soar. In *International Conference on Cognitive Modelling*, 2004.
10. E. Norling. Capturing the Quake Player: Using a BDI Agent to Model Human Behaviour. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1080–1081, 2003.
11. S. Priesterjahn, A. Goebels, and A. Weimer. Stigmergetic Communication for Cooperative Agent Routing in Virtual Environments. In *Proceedings of the International Conference on Artificial Intelligence and the Simulation of Behaviour*, Apr. 2005.
12. H.-P. Schwefel. *Evolution and Optimum Seeking.* Sixth-Generation Computer Technology. Wiley Interscience, New York, 1995.
13. C. Thurau, C. Bauckhage, and G. Sagerer. Imitation learning at all levels of game-AI. In *Proceedings of the International Conference on Computer Games, Artificial Intelligence,Design and Education*, pages 402–408, 2004.
14. C. Thurau, C. Bauckhage, and G. Sagerer. Learning Human-Like Movement Behavior for Computer Games. In *Proceedings of the 8th International Conference on the Simulation of Adaptive Behavior (SAB'04)*, 2004.
15. C. Thurau, C. Bauckhauge, and G. Sagerer. Combining Self Organizing Maps and Multilayer Perceptrons to Learn Bot-Behavior for a Commercial Game. In *Proceedings of the GAME-ON'03 Conference*, pages 119–123, 2003.