Evolution of Human-Competitive Agents in Modern Computer Games

Steffen Priesterjahn, Oliver Kramer, Alexander Weimer and Andreas Goebels

Abstract— **Modern computer games have become far more sophisticated than their ancestors. In this process the requirements to the intelligence of artificial gaming characters have become more and more complex. This paper describes an approach to evolve human-competitive artificial players for modern computer games. The agents are evolved from scratch and successfully learn how to survive and defend themselves in the game. Agents trained with standard evolution against a training partner and agents trained by coevolution are presented. Both types of agents were able to defeat or even to dominate the original agents supplied by the game. Furthermore, we have made a detailed analysis of the obtained results to gain more insight into the resulting agents.**

I. Introduction

The area of commercial computer games has seen many advancements in recent years. The games have become more sophisticated, realistic and team-oriented. At the same time they have become modifiable and are even republished open source. However, there has been only little advancement in the intelligence of the artificial players. They still mostly use hard-coded and scripted behaviours, which are executed when some special action by the player occurs. Instead of investing into more intelligent opponents or teammates the game industry has concentrated on multi player games in which several humans play with or against each other. By doing this the gameplay of such games has become even more complex by introducing cooperation and coordination of multiple players. Thus, making it even more challenging to develop artificial characters for such games, because they have to play on the same level and be human-competitive without outnumbering the human players.

Therefore, modern computer games offer interesting and challenging problems for artificial intelligence research. They feature dynamic, virtual environments and very graphic representations which do not bear the problems of real world applications but still have a high practical importance. What makes computer games even more interesting is the fact that humans and artificial players interact in the same environment. It is possible to play directly against the results of a learning algorithm. Furthermore, data on the behaviour of human players can be collected and analysed.

Steffen Priesterjahn and Alexander Weimer are with the Department of Computer Science, University of Paderborn, 33098 Paderborn, Germany (phone: +49 5251 603345; fax: +49 5251 603338; email: spriesterjahn@upb.de).

Oliver Kramer and Andreas Goebels are with the International Graduate School on Dynamic Intelligent Systems, University of Paderborn, 33098 Paderborn, Germany (phone: +49 5251 603349; fax: +49 5251 603338; email: okramer@upb.de).

This paper presents an approach to evolve artificial players to successfully compete in combat situations in a three-dimensional action game. The artificial players, called agents or bots[1], work on the basis of input/output-rules which are learned by an evolutionary algorithm. We used the game Quake3[2] as the basis of our experiments, because it features dynamic, multi player gameplay. So, it complies to the features discussed above. It is also open source and runs on Windows and Linux, which makes it easy to modify the game. Figure 1 shows a scene of Quake3.

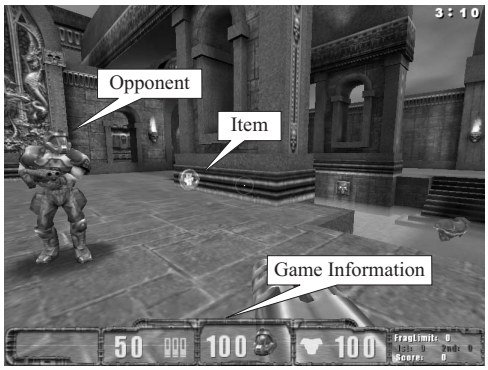


Fig. 1. A Scene from Quake3

This paper is structured as follows. Section II gives an overview of some related work to the topic. The following section describes the basic modelling of our approach, i.e. how the behaviour is encoded and how the evolutionary algorithm is composed. Then, section IV gives information on how we set up the experiments to evaluate our approach. It also describes which parameters we chose to analyse and why we chose certain values for the other parameters. In section V the results of our approach are presented. In these experiments we were able to evolve very competitive agents, which could defeat the standard Quake3-Bot on any difficulty setting. Though, we had to depend on the hard-coded Quake3-Bot as a training partner. So, we also made some experiments using coevolution. These experiments and their results are presented in section VI. Finally, section VII contains a detailed statistical analysis of the obtained results to get more information about the structure of the evolved behaviour.

[1] An abbreviation of robot

[2] id software, 1999

II. Related Work

Modern computer games are more and more frequently used in artificial intelligence research. Especially in the last years the interest in learning and intelligent agents for such games has grown higher and higher.

An interesting approach for such an agent has been proposed by Laird et al. [5]. Their agents try to anticipate the actions of the other players by evaluating what their own planning mechanism would do, if they were in the same position. In a later version reinforcement learning was added to their approach [8]. 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 [6], [7] has also implemented an agent which uses planning and local search. Another interesting approach has been applied by Norling [9], in which a BDI-model (Belief-Desire-Intention) is used to model a human-like agent.

Some very interesting and promising research has been proposed by Thurau et al. [13], [12]. They have produced agents which try to learn some desirable behaviour based on the imitation of other players. In [13] neural nets which are trained on data gained from human players and in [12] neural gas to represent the structure of the environment are used to learn gaming behaviour. Priesterjahn et al. [10] have also introduced an approach to imitate arbitrary players. First the behaviour of the players is recorded in the form of input/output-rules. Then, an evolutionary algorithm is used to sort out the most important rules. The underlying definition of rules and some setup decisions from [10] also apply to this paper. Though, this paper concentrates on the generation of successful behaviours from scratch. In 2004 Cole et al. [3] have introduced a GA which tunes the parameters of agents in the CounterStrike game. In contrast to this paper, they tried to optimise the overall criteria when a certain behaviour should be used, instead of optimising the behaviours itself. In a similar manner Bakkes et al. [1] have used evolutionary algorithms to evolve team strategies for the "capture the flag" game.

Stanley et al. [11] also trained agents for a game-like environment by using neuroevolution. In their system, a human player can build training scenarios to have the agents learn some desirable behaviour. Their learning algorithm is fast enough to work online. However, their focus is more on making a game out of the training process then to get agents on a human-competitive level.

Concerning the underlying evolutionary process, we use an evolutionary algorithm which is based on evolution strategies as described in [2].

III. Basic Modelling

In the following we will give a short description of how our approach is modelled and define the most important terms. Our agent senses its direct environment by dividing it into a *grid* of quadratic regions. The agent is always in the centre of this grid and the grid is always rotated in relation to the agent. So, every grid field has always the same relative position to the agent. Traces are used to see, if a grid field is filled or empty. Each trace sends a ray from the head of the agent to the centre of the respective grid field. If the trace reaches its destination, the field is marked as empty, otherwise it is marked as filled. If an opponent occupies a grid field, the respective field is marked with another special value. The size of the grid is finite and covers only the vicinity of an agent.

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

$$g_{i,j} = \begin{cases} 0, & \textit{if the field is occupied} \\ 1, & \textit{if the field is empty} \\ 20, & \textit{if the field contains an opponent.}^{3} \end{cases}$$

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

So, each grid encodes some special situation in the game, depending on the current environement and the current position of an opponent, if it is in the vicinity.

Each agent has a list of *rules* in its memory. These rules consist of a grid and a command. The *command* just represents an action in the game, e.g. "run forward" and "look left". A detailed and formal description of these terms is given in the following definition.

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.

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

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

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

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

The behaviour of the agents is based on a simple operating cycle as it is shown in figure 2. After the agent has acquired its current grid, it compares this grid to the grids of the rules in its memory. For this comparison, both grids are smoothed with a Gaussian filter [4] and the Euclidean distance between

[3] A value of 20 has been chosen to emphasise the positions of the opponents. The choice of this value was the result of an empiric process.

[4] The Gaussian filter makes it possible to calculate the rough similarity between two matrices. For example, if you compute the Euclidean distance between $(1, 0, 0)$, $(0, 1, 0)$ and $(0, 0, 1)$, they will all be equidistant. However, in our sense $(1, 0, 0)$ and $(0, 1, 0)$ are more similar than $(1, 0, 0)$ and $(0, 0, 1)$. After being smoothed with a Gaussian filter you get for example the vectors $(1, 0.2, 0)$, $(0.2, 1, 0.2)$ and $(0, 0.2, 1)$. Now, the Euclidean distances give a better representation of the similarities between the vectors.

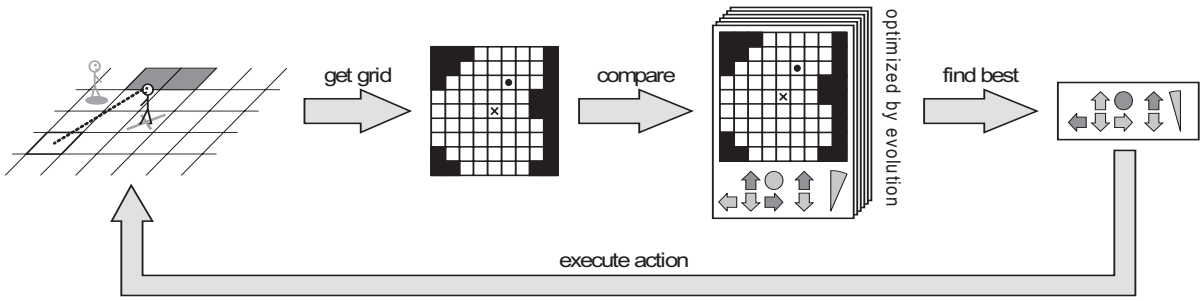


Fig. 2. The Basic Operating Cycle. (x - player, • - opponent)

the resulting matrices is calculated. The rule which contains the best fitting grid is then executed. This process is repeated 10 times per second. So, each time frame for choosing an action has a length of 100 milliseconds.

During the optimisation phase of our approach these lists of rules are optimised with an evolutionary algorithm (EA). Each bot owns a rule list $\{R_1, ..., R_k\} \in \mathcal{R}^k$ with a fixed size $k \in \mathbb{N}$. At the beginning, the first individuals are initialised with randomised rules. That means that we initialise the grid randomly with filled or empty fields and randomly put one opponent on some position on some of the grids. The commands are also randomly chosen. The yaw angle is initialised with a random angle in $[-90°, 90°]$. Table I gives an overview of how randomised rules are constructed.

TABLE I
CONSTRUCTION OF RANDOMISED RULES

Value	Randomly chosen from
grid field	$\{0, 1\}$
opponent on grid	$\{\text{TRUE}, \text{FALSE}\}$
opponent position	random grid field
f	$\{-1, 0, 1\}$
r	$\{-1, 0, 1\}$
φ	$[-90°, 90°]$
a	$\{0, 1\}$

After the initialisation, crossover and mutation are used to select the best rules and to gain further optimisation of the performance of the agents.

Population and Structure:
Concerning the population structure and the selection scheme of our evolutionary algorithm we use a $(\mu + \lambda)$-EA based on 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 lists of the parents. Then, the rule list of an offspring $\{O_1, ..., O_k\} \in \mathcal{R}^k$ is created by randomly choosing each rule O_i from $\{R_i, R'_i\}$ with uniform distribution. So, crossover effects the structure of the rule lists.

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 cannot be moved beyond the grid borders. For the command (f, r, a, φ) of a rule, f, r and a can be set to one of their possible values. The alteration of the view angle φ can be changed by adding a random angle $\Delta\varphi \in [-\alpha°, \alpha°]$.

Simulation and Fitness Calculation:
The fitness of each bot is evaluated by letting it play against the built-in Quake3-Bot and apply its list of rules for a fixed simulation period. The summed health loss of the opponents $h_{\text{opp}} \in \mathbb{N}_0$ and the health loss of the bot $h_{\text{own}} \in \mathbb{N}_0$ are 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. The weights w_{own} and w_{opp} determine the influence of each value.

We noticed that a fitness calculation by $f = h_{\text{opp}} - h_{\text{own}}$ (with $w_{\text{own}} = w_{\text{opp}} = 1$) could lead to an undesirable gaming behaviour. Some agents specialised themselves in running away from the opponent. However, when we chose $w_{\text{own}} = 1$ and $w_{\text{opp}} = 2$ we created bots which tended to behave suicidal. Therefore, we introduced a dynamic fitness calculation. At the beginning, we start with a rather high value for w_{opp}. After each generation, a regression factor $0 < q < 1$ is applied to w_{opp} until w_{opp} reaches 1.0. w_{own} is not changed and set to 1.

IV. EXPERIMENTAL SETUP

Since we were more interested in the capabilities of our approach than on the evolutionary algorithm itself, our interest was aimed at the influence of the design parameters, e.g. the size of the grid or the rule lists, and not on the

parameters of the EA. Therefore, we decided to conduct several experiments concerning the design parameters and to leave the EA-parameters fixed. As a result of an empiric process, i.e. extensive testing and several experiments, we chose those parameters according to table II.

TABLE II
PARAMETER SETUP

Parameter	Value
Population Size $\mu + \lambda$	60
μ	10
λ	50
Mutation Probability p_m	0.1
Yaw Angle Mutation Range α	5°
Evaluation Timespan	60 seconds per agent (1 hour per generation)
w_{own}	1
w_{opp}	starts at 2
Regression Factor q	0.98
Termination	after 3 days (= 72 generations)
Number of Runs per Experiment	5

Each experiment was repeated 5 times to gain statistically more valid results. All experiments were run for three days (72 generations). We have also experimented with some longer runs, but we saw only marginal performance improvements.

Our experiments took place on a small map/level which consisted of only one room and a central column. This was done to increase the probability that the agents actually meet each other. So, the evaluation time could be decreased. Furthermore, since we were only interested in learning the fighting behaviour, a small map was sufficient for our experiments. A Quake3-Bot was placed on this map and played against all agents of the population, one after another. Thus, we had an opponent which plays the same way all the time. This reduces the variations in the fitness function. Though, performance measuring is still influenced by coincidence, e.g. if the bots directly see each other at the beginning of a round or not. We figured out that we need at least one minute of playing time to get reliable results, especially in the first generations in which our agents only show very random behaviour. An even shorter evaluation time would lead to too much fluctuations in the fitness evaluation. On the other side, a too long evaluation time would lead to an even longer running time of the algorithm. The regression factor was chosen so that w_{opp} reaches 1.0 after the first 30 generations.

For the design parameters we again conducted a series of tests to find out good values. We then took the best values and systematically changed each parameter to detect its influence. Table III gives an overview of the conducted experiments. Experiment 1 denotes the base experiment.

It is important to notice that we modified the density and not the size of the grid. So, all experiments were run with a grid of approximately 15 metres x 15 metres in the virtual world. However, in experiments 5 and 6 the grid was separated into 11x11 and 21x21 fields respectively.

TABLE III
EXPERIMENTAL SETUP

#	Grid Size	Rule List Size
1 (base)	*15x15*	*100*
2	15x15	50
3	15x15	10
4	15x15	400
5	11x11	100
6	21x21	100

V. RESULTS

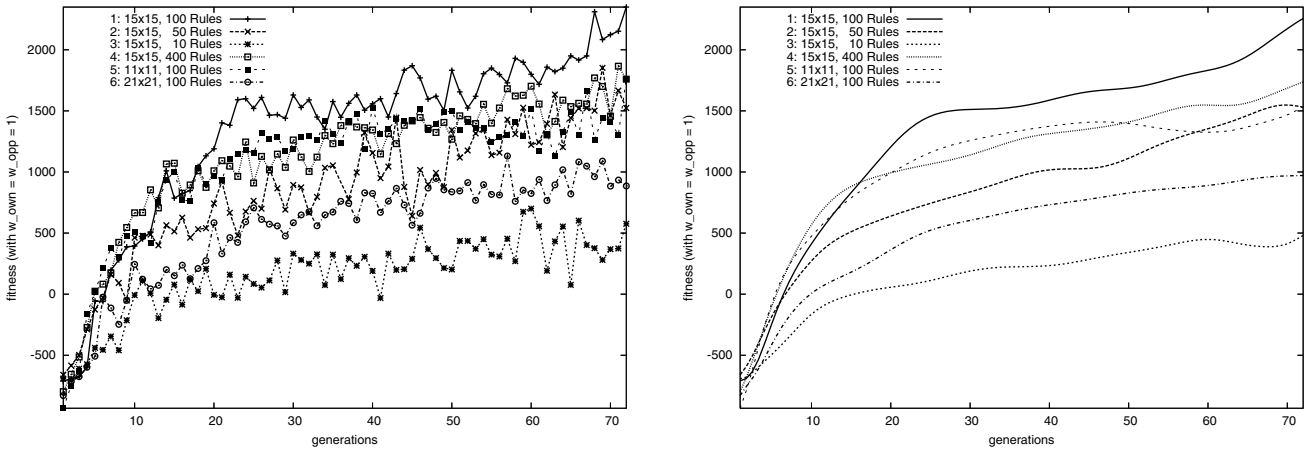
Figure 3(a) shows the results of our experiments. There, the mean of the fitness values of the best individuals of each generation is plotted. For better visibility we plotted the fitness values as if w_{own} and w_{opp} would have been 1 already at the beginning. It can be clearly seen that our agents learned to defeat or to be as good as the Quake3-Bot in all experiments, since they all reached fitness values above zero. In the case of experiment 1 even the mean fitness of all individuals of a generation rose above zero (see figures 3(c) and 3(d)). The best individuals outperformed their opponent already after five to seven generations.

Figure 3(b) shows the same plots as 3(a) smoothed with a Bezier curve for better visibility. Experiment 1 reached the highest performance. In one minute the best agent applied up to 3400 points more damage to the Quake3-Bot than the Quake3-Bot applied to them. This is a very large difference, given that the used weapon is only able to apply up to approximately 100 points of damage per second in the case of a direct hit. So, it can be said that the best evolved agents dominated the hard-coded Quake3-Bots.

Looking at the differences, it can be seen that the size of the rule list has a profound influence on the performance of our approach. Reducing the rule list size from 100 to 50 and 10 reduces the performance. Using a rule list size of only 10 rules results in the worst performance of all experiments. However, it is not the case that using a larger rule list always results in a better performance. Experiment 4, which used 400 rules per agent, performs worse than the base experiment.

Concerning the grid densities, the experiments show that a too dense (experiment 6) or a too sparse grid (experiment 5) can compromise the performance. In the case of the sparse grid details might get lost, which affects the decision making or rule selection process. On the other side, a too dense grid blows up the search space, which compromises the convergence speed of the underlying EA.

In addition to the consideration of the pure fitness development we think that it is even more important to assess the gaming behaviour of the evolved agents on a quality level. Though, such an assessment can only be very subjective. As it is common for computer games, we call a behaviour a good



(a) Mean of the *best* Individuals in each Generation

(b) Smoothed Plots of Figure 3(a)

(c) Mean of *all* Individuals in each Generation

(d) Smoothed Plots of Figure 3(c)

Fig. 3. Experimental Results

gaming behaviour if it looks fluid and generally speaking human-like.

Our agents showed a very aggressive behaviour and were able to move fluidly. [5] Interestingly, almost all experiments resulted in agents which tried to hunt and closely follow their opponent. At the same time they tried to avoid the attacks of their opponent by running from one side to the other. Playing against them is quite hard, because they really put the player into a defensive position.

As mentioned above, if we do not use the dynamic fitness function adjustment and simply use $f = h_{\text{opp}} - h_{\text{own}}$ as a fitness function, the agents will learn to run away from the opponent in some experiments. So, running away seems to be a local optimum. It minimises the own health loss. Once caught in this behaviour, it is not easy to learn that the fitness can even be further increased when the agent attacks its opponent, because it would first mean a deterioration of the performance when the behaviour is changed. Therefore, the aggressive behaviour shown by the best agents in our experiments might also be only a local optimum. However, since this behaviour showed up in almost all experiments and in always all setups and given the reached performance, it is also a very good local optimum.

VI. Coevolution

Having seen that our approach is able to create successful behaviour from scratch, we wanted to see if it would be possible to work without a third party opponent – namely the Quake3-Bot – to measure the performance of the agents. In practice we cannot assume that we have any hard-coded agents at our disposal to use them as training partner. Therefore, we decided to evolve gaming agents using coevolution.

We simply took two populations which used the same parameters as in the base experiment above. These populations were synchronised so that the n'th agent of population one would always play against the n'th agent of population two. Since coevolution normally needs more time to converge we granted the algorithm a significantly longer running time of 200 generations.[6].

After that process we took the final generation and evaluated their performance by letting each of them play for one

[5] See www.upb.de/cs/ag-klbue/de/staff/spriesterjahn/videos/evobot.avi for a demonstration.

[6] This resulted in a running time of more than one week.

minute against the Quake3-Bot. As a result this approach was able to produce agents which could perform as good as the Quake3-Bot. Some defeated it by a margin of up to 1000 health points. This shows that well performing results can be produced by coevolution.

Though, the behaviour of the evolved agents was not as fluid as the behaviour of the agents which were evolved by standard evolution. They moved a bit choppy and therefore were easily identifiable as artificial players. This behaviour is similar to the behaviour of the agents we obtained by standard evolution in an early stage. So, we made a longer run of more than 300 generations. The behaviour of these agents was indeed more fluent. However, the performance improvement over using 200 generations was only marginal.

VII. Analysis of the Results

This section presents an analysis of the evolved rule lists, to find out more about the structure of the gained rules. We applied a statistical analysis to the rule selection behaviour of the agents. We chose to use second order statistics, because we were interested in the relations between the single rules. Therefore, we used co-occurence matrices as defined in the following definition.

Definition 3 (Co-occurrence Matrix)
For a rule list $\{R_1, ..., R_n\} \in \mathcal{R}^n$, the Co-occurrence Matrix P is defined as $P = (p_{i,j})_{1 \leq i,j \leq n}$, where $p_{i,j}$ denotes the probability that rule R_j is executed directly after the execution of R_i.

We were especially interested if the rules are executed in repetition or if there is a structure in the rule lists in which a row of special rules is executed. Therefore, we defined the following admeasurements.

Definition 4 (Reflexivity ρ, Transitivity τ)
For a given co-occurence matrix $P = (p_{i,j})_{1 \leq i,j \leq n}$, the value $\rho \in [0, 1]$ whith

$$\rho = \sum_{i=1}^{n} p_{i,i}$$

is called the reflexivity of P. The value $\tau \in [0, 1]$ whith

$$\tau = 1 - \rho$$

is called the transitivity of P.

The reflexivity ρ indicates the strength of the main diagonal of the matrix and denotes the overall probability that rules are executed in repetition. The transitivity τ denotes the probability that another rule is chosen after one rule has been executed.

We took the best performing agents from each setup and computed the respective values after they had played for a longer time period[7]. We also did some first order statistics and calculated the standard deviation σ for the probability of a rule to be selected. A low standard deviation would indicate that the rules are executed rather uniformly distributed. A higher value would indicate that there are big differences between the execution counts of the single rules, e.g. when only five out of one hundred rules are really used. The results are presented in table IV. We also added the reflexivity and transitivity of a rule list which we gained by the imitation based approach in [10].

TABLE IV
Statistical Analysis

#	Grid Size	Rule List Size	σ	ρ	τ
1	15x15	100	0.34	28%	72%
2	15x15	50	0.24	30%	70%
3	15x15	10	0.28	36%	64%
4	15x15	400	0.20	26%	74%
5	11x11	100	0.23	24%	76%
6	21x21	100	0.21	24%	76%
Coevolution	15x15	100	0.20	23%	77%
Imitation	25x25	50	0.06	31%	69%

The values for ρ and τ are relatively similar in all experiments, including coevolution. This seems to lead to the conclusion that all the best rule lists are organised in a similar way. Interestingly the values from the imitation based rule list are also very similar to the other ones except the standard deviation. This indicates that there is a similar structure in the imitation based rule list, but the work is distributed onto a higher number of important rules.

For a further analysis we chose to look at the co-occurence matrices themselves. Figure 4 shows some samples of the gained matrices. The x- and y-axis denote the number of a rule in the respective rule list. The z-axis stands for the probability that rule i is followed by rule j.

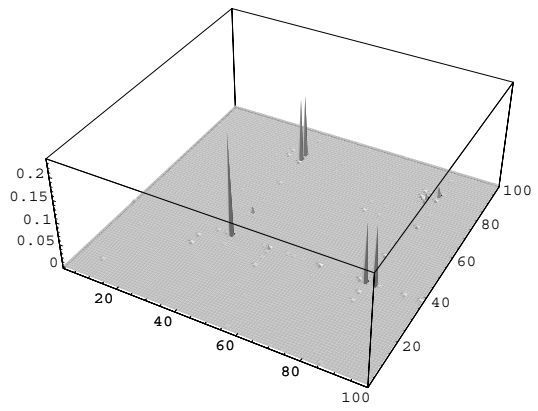
Figure 4(a) shows the co-occurence matrix of the best individual that could be evolved. It originates from the base experiment using 100 rules and a 15x15 grid. It shows a structure that we have found in most of the high performing agents. It consists of just one main rule[8] which is executed in repetition and in this case two supporting rules[9] which are mostly only executed once and then a switchback to the main rule occurs.

To gain more insight into this structure table V shows the transition probabilities of the three most important rules in figure 4(a). 23% of all transistions are transistions from rule 37 to 37. So, in almost one quarter of all rule executions this rule is executed in repetition. Therefore, rule 37 is the main rule. A transition from the main rule to the supporting rules 82 and 85 occurs in 14% of all transitions respectively. The same holds for transitions from these rules to the main rule. However, transitions between the supporting rules and repetitions of these rules occur only rarely. So, in most cases the main rule is executed right after the supporting rules have been executed just one time. They are just used to correct

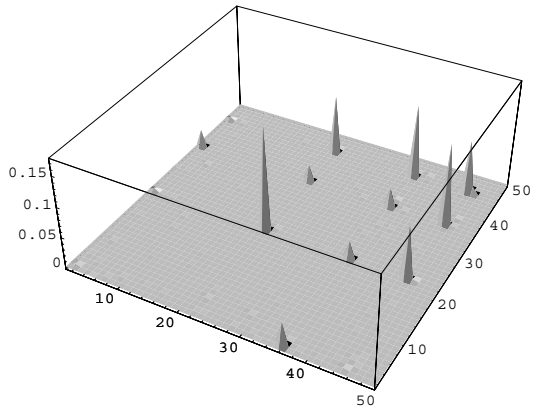
[7]At least 20 minutes

[8]The peak on the main diagonal at position (37,37)

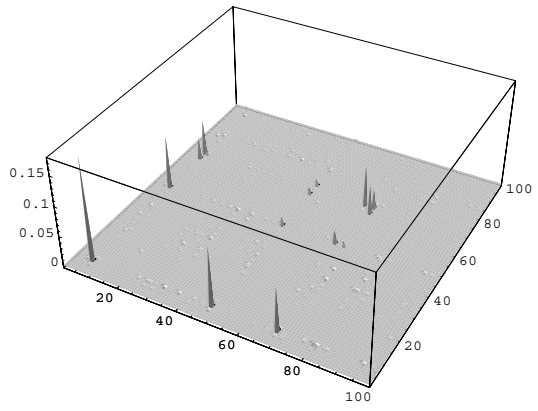
[9]The symmetric peaks at (37,82), (82,37), (37,85) and (85,37)



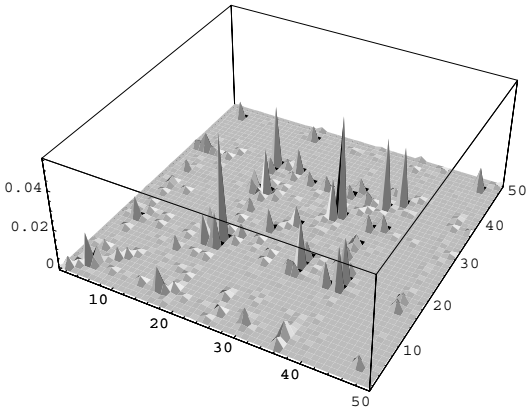
(a) 15x15, 100 Rules



(b) 15x15, 50 Rules



(c) Coevolution



(d) Based on Imitation

Fig. 4. Some Co-occurrence Matrices (The Plots have different Scales.)

certain actions. Though, without them the agent would not be as successful as it is. Another point that should be noticed is that the sum of the transition probabilities of these three rules is just about 80%. So, in 20% of all cases transitions between the other rules occur. As figure 4(a) shows, these transistions are distributed very evenly and range between 0% and 1%. So, the other rules still have an influence, although they are used quite rarely. Some of them might encode behaviours for some special situations, e.g. if the agent has run into a corner.

TABLE V
THE HIGHEST TRANSITION PROBABILITIES OF FIG. 4(A)

rule #	37	82	85
37	23%	14%	14%
82	14%	1%	0.5%
85	14%	0.2%	2%

The main rule encodes the main behaviour of the agent, whereas the supporting rules correct this behaviour to adapt to the behaviour of the opponent. So, the overall behaviour is encoded in the interplay of these three rules. Other rule lists with a similar structure use three or four supporting rules or show a slight modification of this schema. For example figure 4(b) shows one main rule and two supporting rules, where one supporting rule is also run in repetition for some periods.

It might be surprising that only 3 of 100 rules are really used. However, already one rule is enough to encode the behaviour to circle around an opponent, which is not a trivial and also a very successful behaviour often shown by human players.

In another experiment we reduced the rule list to just these three rules. The agent which used this rule list was following and attacking the opponent. So theses rules were responsible for that behaviour. However, the agent got into problems when it went into a corner or could not see the opponent. So, the other rules were indeed important to handle such situations, as we suspected above. When we allowed the agent to use all rules which were executed more often than the mean of all rules (11 of 100 rules), the agent was able to show almost the same behaviour as with the full rule list.

Another interesting point is that using a large rule list generates such a better performance, though only few rules are really needed. We think that this is caused by the fact

that at the beginning a much broader base of different rules to draw from is generated when using larger rule lists. Furthermore, a crossover operation on larger rule lists also has a higher probability to draw good rules from the parents, because the position of the rules is fixed. In small rule lists there is a higher probability that two good rules are on the same position, whereby only one of them can be chosen in a crossover operation. The impact of this effect is decreased once the rule lists are large enough and other effects begin to deteriorate the performance. So, a large number of rules per individual kind of improves exploration without damaging exploitation. If the number is too large, it will blow up the search space and can also lead to delays, because the agent has to look up too many rules in each time frame.

Interestingly, coevolution also produces results which fall into the same schema. Figure 4(c) shows the co-occurence matrix of the best individual that was obtained by coevolution. This indicates that coevolution can find similar solutions but only needs more time to find them.

For comparison figure 4(d) shows the co-occurence matrix of the best agent which we produced by the imitation based approach in [10]. There is some significant difference, as the evaluation of the standard deviation already indicated above. Much more rules are used and there is a bunch of special rules for special events and behaviours. This agent was also able to show more complex behaviour. For example it was able to take cover behind a column. We could not reproduce this behaviour with the approach presented in this paper, even when making some longer running experiments. However, the performance of the best agents from the approach in this paper was on the same level as the performance of the imitation based agents.

VIII. Conclusion & Future Work

We have presented an approach to successfully evolve agents for modern computer games. The agents were not only able to play as good as the provided hard-coded agent, they were even able to dominate it on any difficulty level. In addition, our approach was also able to produce competitive agents already after few generations. We also tested the best trained agents on larger maps. They were still successful in close combat situations, as we intended. So, the results can be also used on larger maps. Furthermore, the used rules can also be easily adjusted for other environments and tasks. Therefore, our results can be adapted to many games and even be used in other problem fields.

Concerning coevolution we have shown that our approach is also able to deliver competitive results without some preprogrammed training partner. Therefore, it can be used to train agents in games and environments which do not yet feature any artificial players.

In a detailed analysis we were able to show that only few rules are sufficient to reach a high performance. We found out that in all high performing experiments the result was a structure in which few rules worked together. In these cases, some rules encode a special behaviour, whereas others correct certain movements and are only executed once at a time. So, the overall behaviour is encoded in the interplay of several rules.

In the future we want to concentrate on developing an approach which is able to learn online. Therefore, we think about removing the generational aspect of this approach. We will also try to evaluate single rules and not whole rule lists. By doing this, we want to reach an acceleration of the convergence rate. This could also lead to an improvement to the usability of coevolution. Several agents could learn advantageous behaviours and share their knowledge with their teammates. Furthermore, we want to concentrate more on the imitation of other players and the cooperation with other agents.

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] N. Cole, S. J. Louis, and C. Miles. Using a genetic algorithm to tune first-person shooter bots. In *Proceedings of the International Congress on Evolutionary Computation*, 2004.

[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] 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.

[6] 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.

[7] 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 *Knstliche Intelligenz 2*, pages 51–53, 2002.

[8] S. Nason and J. Laird. Soar-RL: Integrating Reinforcement Learning with Soar. In *International Conference on Cognitive Modelling*, 2004.

[9] 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.

[10] S. Priesterjahn, O. Kramer, A. Weimer, and A. Goebels. Evolution of reactive rules in multi player computer games based on imitation. In *Proceedings of the International Conference on Natural Computation*, 2005.

[11] K. O. Stanley, B. D. Bryant, , and R. Miikkulainen. Evolving neural network agents in the nero video game. In *Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games (CIG'05)*, 2005.

[12] 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.

[13] 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.