STUDIA INFORMATICA
Nr 2(9) Systemy i technologie informacyjne 2007

Automatic Strategies for Autonomous Virtual Characters

Lach Ewa[1]
[1] Institute of Informatics, Silesian University of Technology,
Akademicka 16, 44-100 Gliwice, Poland

Abstract. Technique presented in the paper concerns automatic generation of strategies controlling virtual characters' behaviour. Virtual people are currently widely used in many applications, especially in computer games, films and educational systems. A lot of researches focus on creating intelligent characters capable of deciding about their actions. The fully acceptable solution has not yet been found. The paper presents the problem of generating strategies by means of modified genetic programming. A new Guide-Path technique is introduced.

Keywords. Genetic programming, task control, machine learning, virtual characters animation

1 Introduction

Animation techniques commonly employed in fields of movies and games rely mainly on motion capture systems and keyframed motion, which are still relatively expensive and time-consuming. The solution to this problem might be a creation of autonomous, aware of their environment, animated characters capable of independent planning of their actions.

The paper presents experiments on using modified layered learning genetic programming (LLGP) to evolve behaviours of virtual characters. Genetic programming (GP) is domain-independent, problem solving approach in which a population of computer programs (individuals) is evolved to find a solution. The simulated evolution in GP is based on the Darwinian principle of reproduction and survival of the fittest ([17]). Layered learning technique consists of breaking the main problem up into a hierarchy of sub-problems, which are solved sequentially, by using results of lower layers in the next layer ([2]). The author of the paper proposed new Guide-Paths (GuP) technique, which allows using information about virtual character's behaviours during GP evolution.

The remaining of the document is organized as follows. Section 2 surveys related work. Section 3 describes Guide-Paths technique. Section 4 and 5 describe experiments and results. Section 6 concludes the paper and discusses the directions for future researches.

2 Related Works

Researches in a generation of intelligent characters are believed to start with Craig Raynolds distributed behaviour model for simulating flocks of birds in which global behaviour arises from the interaction of simple local rules [11]. Each individual member of the flock has a simple rule set stating that it should move with its neighbours while avoiding collisions with them. The model proposed by Raynolds was able to create a realistic behaviour of flocks of birds, but was not capable of creating motions of a single bird. To achieve it an increase of rules was proposed. In [6] 5200 rules were used to control behaviour of a pilot. The result was remarkable, but manual creation of so many rules for each new behaviour for several characters is not acceptable. A different approach to the problem was proposed by Aaron and his colleagues. In [1] they presented a methodology based on the integration of nonlinear dynamical systems and kinetic data structures. Carefully designed attractor and repeller functions for targets and obstacles, shaped with differential equations, modelled the behaviour of an autonomous agent. However, this solution seems unsuitable for complex characters in environment with many objects and agents. Terzopoulos and Tu [15] presented a system for animating dynamically simulated fish. The intentions of the fish were evaluated based on not only an environment state but also on the fish's habits and state of mind. Ulicny and Thalmann [14] proposed a model for crowd simulation based on combination of rules and finite state machines for controlling agents' behaviour in a multi-layer approach. At higher levels the rules selected complex behaviours based on agents and environment states. Hierarchical finite state machines implemented lower level complex behaviours. Each behaviour was controlled by one finite state machine. In another approach [7] agents evaluated, on each iteration, hierarchical decision graph. States of the virtual human and environment's data were taken into account during the action selection.

A manual creation of proposed structures can be problematic in presented methods. Any enhancements to character behaviour repertoire usually require time and expert knowledge from an animator. As a result researchers started to explore methods which enable virtual characters to learn for specific assigned tasks. Additional benefit of using learned behaviours is the fact that they are, in some cases, more general and could be quite innovative.

The most promising techniques for automatic creation of complex behaviours of virtual characters are evolution programming and reinforcement learning (RL). Tang and Wan proposed genetic algorithms (GA) to simulate intelligent self-learning characters [13]. GA was also used to produce binary rules, which defined behaviour of autonomous players in a virtual soccer game [4]. Gustafson and Hsu [5] applied layered learning genetic programming to evolve behaviour of team of agents playing a keep-away soccer, a game where a team of four players had to prevent a single opposition player for coming into contact with the ball. A keep-away soccer was also addressed by Kohl [16], whose players were controlled by neural networks evolved by means of GA. Bajurnow and Ciesielski [2] studied performance of LLGP for evolving goal scoring behaviour of soccer players.

Blumberg [3] proposed learning motion in real-time during the interactions with the system, which used so-called pose-graph to generate motion. The nodes of the graph were derived from source animation amended by an interpolation technique. Thus the animation was realistic and transitions could be generated in real-time but the actions had to be prepared by an animator and pre-programmed into system. Szarowicz in [10] applied updated Q-learning algorithm to automatically acquire action sequences for animated characters. In another approach [12] the technique inspired by RL was used for automatic generations of scripts controlling behaviour of fighting characters. Before each fight new script was generated from rules with wages calculated according to winnings and losses of characters controlled by scripts.

Presented researches in an automatic creation of behaviours of virtual characters produced interesting results, but the fully acceptable solution has not yet been found. Proposed in the paper new Guide-Paths technique with LLGP allows a generation of behaviours at different levels of abstraction. To change behaviour of a virtual character an animator has to only change character's goals. A new solution needs improvement, but even its early version shows great potential to become answer to many animators' problems.

3 Guide-Paths Technique

Behavioural strategies, described in this paper, are understood as sequences of operations run successively or in parallel according to character's states or states of his environment to obtain character's goals. Strategy trees (Figure 3.1), formalizing behavioural strategies, are ordered trees processed from root to leaves with nodes representing operations. Operations are responsible for performing transformations on three-dimensional objects and determining environment's and virtual characters' states. Operations can form new complex operations: strategy trees. Strategy trees can be created manually, but for complicated tasks the process can become time-consuming and prone to errors. Because of that the author employed layered learning genetic programming for automatic strategies generation [8]. Results were very promising. In order to improve them new extension to genetic programming in the form of Guide-Paths technique was designed.

The simulated evolution in GP starts with an initial population of strategy trees, generated at random. Then, each strategy tree is measured in terms of how well a virtual character performs in its environment. This measure is called a fitness measure. Genetic programming is guided by the pressure exerted by the fitness measure and natural selection. This fact makes choosing an adequate fitness function one of the most important task, when using genetic programming. Fitness measure describes virtual character's goals. For example, in a situation of a virtual human approaching a door the fitness function could measure a distance between the human and the door. The closer to the door the human is, the more fitted the strategy tree is. Typically, each strategy tree is run over a number of fitness cases representing different situations (for example, initial positions of the virtual human). Usually strategy trees in initial generations have a very poor fitness. Nonetheless, some trees in a population are more fitted than the others and these differences in performance are exploited by genetic programming. Iteratively, strategy trees are evaluated for fitness and genetic operations are performed on those trees to generate new populations. Over many generations, strategy trees tend to exhibit increasing average fitness.

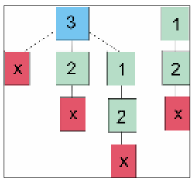

Figure 3.1. Two exemplary strategy trees. Numbers are identifiers of operations.

In LLGP a bottom up decomposition of goals is applied. Virtual characters first learn simpler tasks, then compose and coordinate them to solve larger tasks. For each layer a different fitness function is used.

During generation of population of strategy trees GuP allows using information about order of operations in strategies. For example, in the situation of a virtual hero trying to pass through enemy territory, when operations were run, which determined that an enemy is approaching, the hero is low on ammunition and there is a rock nearby, guiding paths could state that with probability equal to 0.65 the hero should hide behind the rock. He could also with probability equal to 0.05 retreat or with probability 0.1 attack the enemy. Correctly constructed guiding paths should discourage the hero from trying to start a conversation with the enemy.

A k-length guiding path consists of a sequence of elements $\{o_1.s_1(p_1)\ o_2.s_2(p_2)\ \ldots\ o_k.s_k(p_k)\}$, where o_i identifies an operation, s_i - its successor number and p_i defines conditional probability of running an operation o_i provided that sequence of operations with given successors $<o_1.s_1,o_2.s_2,\ldots,o_{i-1}.s_{i-1}>$ was run for $i = 2,3, \ldots,k$. p_1 defines probability of running an operation o_1, when previously run operations are not considered. Successor number points out which an alternative subpath is chosen when more than one is given.

In our example, with the hero and his enemy, we can identify three guiding paths:
{1.1(0.2) 2.4(0.35) 3.2(0.02) 4.1(0.65) ...},
{1.1(0.2) 2.4(0.35) 3.2(0.02) 5.1(0.05) ...},
{1.1(0.2) 2.4(0.35) 3.2(0.02) 6.1(0.10) ...},
where an operation determining the position of the enemy has identifier equal to 1 and its first successor (s_1=1) is called when the enemy is approaching. o_2=2 identifies an operation verifying amount of an ammunition possessed by the hero and its fourth successor is called when the hero is low on it. Other operations test the position of the rock (o_3=3) and execute hiding the hero (o_4=4), its retreat (o_4=5) or attack (o_4=6).

3.1 Guiding Paths Generation

Guiding paths can be created manually with an expert's help. An expert can formulate guiding paths directly or by designing strategy trees, from which guiding paths can be extracted. A second way, which does not demand estimating probability for each operation in a sequence is simpler, but still requires a knowledge on patterns controlling virtual people behaviours for different tasks. Both methods need an expert's time and focus. Sometimes the patterns are not obvious - especially when there are many low-level operations. Therefore the author proposed an automatic method for strategy trees generation consisting of following steps:

1. Run LLGP evolution r times.
2. Choose at most $r \times z_{max}$ strategy trees from ones generated in step 1 and belonging to one of following sets:
   - strategy trees with fitness function value greater then f_{min},
   - strategy trees with more hits (found solutions for different initial conditions) then h_{min},
   - strategy trees with the best fitness functions values in a run,
   - strategy trees with the greatest number of hits in a run.

Values of variables: r, z_{max}, f_{min}, h_{min}, are determined by an animator.

Acquired trees can be used to extract guiding paths. It can be achieved in the following steps:

1. Define paths (which appear in processed trees) in the form of sequence of elements $<o_1.s_1,o_2.s_2,...,o_j.s_j>$, where o_i identifies an operation, s_i - its successor number, i=1,2,...,j and j defines path length.
2. Define k-length permutations with repetition of existing operations with successor numbers $<o_1.s_1,o_2.s_2,...,o_k.s_k>$ and compute for each $o_i.s_i$: l_i, which defines how many times path $<o_1.s_1,o_2.s_2,...,o_i.s_i>$ appeared in paths from step 1.
3. Remove from permutations of step 2 $o_i.s_i$ with $l_i = 0$.
4. Compute l value as a sum of l_1 of all unique $o_1.s_1$.
5. Compute for each $o_i.s_i$: $p_{i(U)}$ using one of equations 3.1 – 3.4. During guiding paths' generation $p_{i(U)}$ is used to remove insignificant subpaths.
6. Remove $o_1.s_1$ with $p_{1(U)} < p_{min1}$.
7. Remove $o_i.s_i$ with $p_{i(U)} < p_{min}$ for $i > 1$.
8. Compute for each $o_i.s_i$: p_i using one of equations 3.1 – 3.4.
9. Write down guiding paths .

Values of variables: k, p_{min1}, p_{min} are determined by an animator.

The author of the paper proposed three ways of computing conditional probability of running an operation o_i for i>1 (eq. 3.1 - 3.3) and two ways of computing p_1 (eq. 3.3, 3.4). All of them are using relations between paths frequencies, observed in processed trees. For example it can be observed in trees from Figure 3.1 that after operation with identifier 1 in 100% cases its first successor is operation with identifier 2.

$$p_i = \frac{l_i}{l_{i-1}} \quad \text{for } i>1 \qquad (3.1)$$

$$p_i = \frac{l_i}{l_1} \quad \text{for } i>1 \qquad (3.2)$$

$$p_i = \frac{l_i}{l} \quad \text{for } i=1,2,\dots,k \qquad (3.3)$$

$$p_1 = \frac{1}{n}\ , \quad \text{where n-number of used operations} \qquad (3.4)$$

The method of strategy trees' creation should be taken into consideration, when choosing one of equations 3.1-3.4. Big trees generated during genetic evolution need removal or limited exploitation of paths, which appearance in trees was incidental. On the other hand, in small trees designed by an expert every operation matters. Equations 3.2 and 3.3 point out paths $<o_1.s_1,\ o_2.s_2,\ ...,\ o_i.s_i>$, which appear rarely in relation to all paths or paths beginning with $o_1.s_1$. When we want to obtain strategy trees following order of operations found in processed trees equation 3.1 is more useful.

A simple example of a guiding paths generation is presented below. Two strategy trees, from Figure 3.1, were selected for a paths' extraction. Nodes' names correspond to operations' identifiers. In chosen trees four paths were found: <3.1, id_L>, <3.2, 2.1, id_L>, <3.3, 1.1, 2.1, id_L>, <1.1, 2.1, id_L>, where id_L represents an ending of a strategy path. For k=3 we can define 216 (6^3) permutations with repetition of set {1.1, 2.1, 3.1, 3.2, 3.3, id_L}. We can remove permutations beginning with id_L. Now, we have $216-6^2=180$ sequences. After calculating how many times each subpath from permutations appeared in four trees' paths, subpaths with $l_i=0$ were removed. Following sequences remained: <3.1 ($l_1=1$), id_L ($l_2=1$)>, <3.2 ($l_1=1$), 2.1 ($l_2=1$), id_L ($l_3=1$)>, <3.3 ($l_1=1$), 1.1 ($l_2=1$), 2.1 ($l_3=1$)>, <1.1 ($l_1=2$), 2.1 ($l_2=2$), id_L ($l_3=2$)>, <2.1 ($l_1=3$), id_L ($l_2=3$)> for l=8.
For such simple strategy trees a subpaths' removal does not make sense, so it was omitted. Equation 3.1 for i>1 and equation 3.3 for i=1 were used to calculate conditional probability p_i of running an operation o_i In the end following five guiding paths were obtained: {3.1(0.125) id_L(1)}, {3.2(0.125) 2.1(1) id_L(1)}, {3.3(0.125) 1.1(1) 2.1(1)}, {1.1(0.250) 2.1(1) id_L(1)}, {2.1(0.375) id_L(1)}.

3.2 Guiding Paths Application

The aim of a Guide-Paths technique is to store and provide patterns characterizing promising virtual characters behaviours. This patterns can be improved or adapted to new situations during genetic evolution. Thanks to them more general solutions can be created. The Guide-Paths technique can be used during GP evolution in a process of choosing a new tree node, finding a node to remove and testing if a node is necessary in a strategy tree.

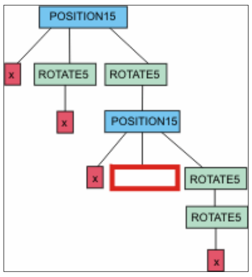

Figure 3.2. Strategy tree with a place (an empty rectangle with wider borders) for a new node

An addition of a new node to a strategy tree consists of finding in the tree a subpath $<v_j, v_{j+1}, ..., v_i>$, where node v_i is a parent of a new node, $j=i-k+2$ for $i \geq k$ or $j=1$ for $i < k$, k is a length of guiding paths. If a sequence of elements equivalent to path $<v_j, v_{j+1}, ..., v_i>$ can't be found in guiding paths then j is increased by 1 and a new sequence is looked for in guiding paths. When the sequence is found a new node is chosen with conditional probability defined by guiding paths. If a sum of probabilities for node v_{i+1} from subpath $<v_j, v_{j+1}, ..., v_i, v_{i+1}>$ is less than 1 then with remain probability random operation is chosen.

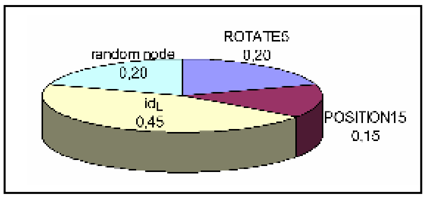

Figure 3.3. Probabilities of choosing different operations for a new node from a tree of Figure 3.2

Figure 3.2 shows strategy tree with operations' names created by merging the name of the operation type and its identifier number. More information on operations classification can be found in [9]. For a tree from Figure 3.2 using following guiding paths: {5.1(0.4) 15.2(0.12) 5.1(0.20) id$_L$(1.0)}, {5.1(0.4) 15.2(0.12) id$_L$(0.45)}, {5.1(0.4) 15.2(0.12) 15.1(0.15) 5.1(0.3)} we can specify probabilities of choosing specific operations for a new node (Figure 3.3): with probability 0.45 the strategy path will end with node v_{i+1}; with probability 0.20 operation 'ROTATE5' will be chosen; with probability 0.15 operation 'POSITION15' will be selected; with probability 0.2 random operation will be chosen.

With guiding paths we can also find a node in a strategy tree for removal. If maximal conditional probability of an operation for its all successors, according to guiding paths, is less than some specific value the operation node can be removed.

The author of the paper proposed following modified genetic operators using Guide-Paths technique to add new nodes and determine possibility of node removal:
- initial population generation (GP-InitGupOp),
- crossover (GP-CrossoverGuPOp),
- standard mutation (GP-MutationStandardGuPOp),
- swap mutation (GP-MutationSwapGuPOp),
- shrink mutation (GP-MutationShrinkGuPOp).

Each genetic operator uses Guide-Paths technique with probability determined by operator's trust coefficient (w_{UB}, w_{UC}, w_{UMstd}, w_{UMswap}, $w_{UMshrink}$), which allow applying guiding paths we do not have full confidence in.

4 Experiments

In order to test the proposed Guide-Paths technique an exemplary learning task, comprising unlocking a door upon touching the door handle, pushing the door and passing through it, was used. Applied virtual human model (avatar) borrows its biomechanical characteristics from robotics. The avatar has a set of joints whose movements can be either prismatic or revolute and are ruled by the forward kinematics. In the conducted experiments an avatar could rotate its arms and forearms (Figure 4.1), walk forward, turn and check environment state or its own position in respect to certain conditions. Table 4.1 presents available types of strategy trees nodes, which detailed description can be found in [9]. To make finding correct strategy trees more difficult given operations include six unnecessary ones. Virtual world consists of a space with a single door and walls.

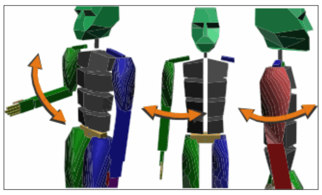

Figure 4.1. Avatars degree of freedom

During decomposition of the main problem two layers were defined. The objective for the first layer was to came within a reaching distance of a door handle from random position and lay a hand on it. Second layer measured how well an avatar fulfilled the main task (passing through the door). The fitness measures are computed as follows:

$$f_1 = \frac{1}{\sqrt{(d_K + 0.6d_D + p_K + p_O)} + 1} \qquad (4.1)$$

$$f_2 = \frac{1}{\sqrt{(d_B + p_K + p_O)} + 1} \qquad (4.2)$$

where:
d_D - distance between an avatar and a place in reaching distance to the handle,
d_K - distance between the hand and the handle,
d_B - distance between the avatar and the place behind the door,
p_K - penalty for too many run operations and collisions,
p_O - penalty for the avatar's wrong facing direction.

Table 4.1. Operations used to build strategy trees

Symbol	Description	Degree
MOVE0	move forward	1
ROTATE1	rotate a right arm forward	1
ROTATE3	rotate a right arm backward	1
ROTATE2	rotate a left arm forward	1
ROTATE4	rotate a left arm backward	1
ROTATE5	rotate a right arm up	1
ROTATE7	rotate a right arm down	1
ROTATE6	rotate a left arm up	1
ROTATE8	rotate a left arm down	1
ROTATE9	straighten right arm in elbow	1
ROTATE11	flex right arm in elbow	1
ROTATE10	straighten left arm in elbow	1
ROTATE12	flex left arm in elbow	1
ROTATE13	turn right	1
ROTATE14	turn left	1
POSITION15	an avatar's position to a door	3
POSITION16, POSITION17, POSITION18	an avatar's hands position to a knob in three dimensions	3
POSITION19	a distance between an avatar and a door	4
ORIENTATION20	avatars orientation to a door	4
IF23	door opened	2

The experiments were performed for GP parameters presented in Table 4.2. An experiments' aim concerned determining whether applying Guide-Paths technique improves learning by means of layered learning genetic programming. The methods compared were standard layered learning genetic programming (LLGP) and LLGP with Guide-Paths (LLGP-GuP), where guiding paths were used with probability 0.9. Guiding paths used in experiments were generated from strategy trees obtained in ten standard LLGP runs with random fitness cases. Their quality was moderate. Strategy trees allowed for avatar's unnecessary movements and did not find solutions for all fitness cases. Twenty runs of each type of experiments were performed.

Table 4.2. Genetic programming parameters

Fitness	eq.4.1, eq.4.2
Population size	250
Generations	2000
Cases	10
p_C	0.9
p_M	0.01

Figure 4.2 shows a few shots from actual animations generated with strategy trees from LLGP-GuP experiments.

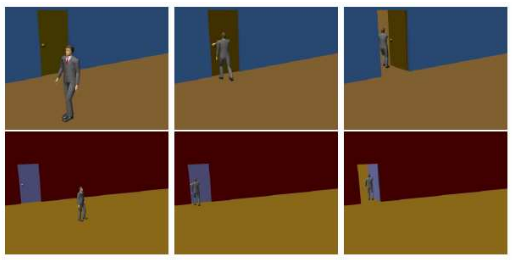

Figure 4.2. Animations derived form exemplary strategy trees

5 Results

Two figures (5.1 and 5.2) present results of conducted experiments. An ordinate, form Figure 5.1 and Figure 5.2, shows percentage of fitness cases, for which strategies found a solution (hits). Figure 5.1 shows best strategies' percentage of hits for each conducted experiment. Results were sorted from worst to best and connected by line for better visualization. Number of experiments, which found strategies with 100% of hits increased by 100% (from 6 to 12) after Guide-Paths application. Strategies with no less then 90% of hits are generated by 90% of LLGP-GuP experiments and 35% of LLGP experiments. It can also be observed that the worst LLGP-GuP experiment generated strategies with 70% of hits, when the worst LLGP experiment generated strategies with 30% of hits. Figure 5.2 shows percentage of hits for succeeding generations calculated as average of experiments best strategies' percentage. We can observe clear difference between speed, with which experiments finds better strategies. An average percentage obtained by strategies of LLGP experiments in 2000 generation was obtained by strategies of LLGP-GuP experiments in 200 generation. Additionally the earliest generation to generate strategy with all hits was 74 for LLGP-GuP and 183 – for LLGP and average generation was 448 and 970.

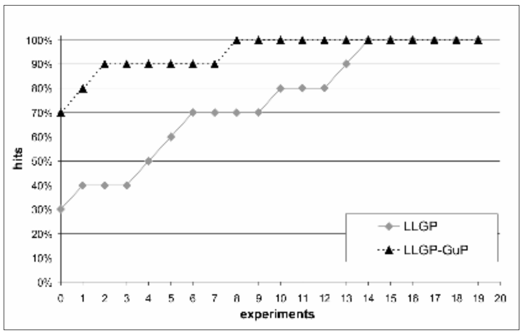

Figure 5.1. The best strategies for each conducted experiment. Comparison of experiments with and without Guide-Paths.

Calculation of generated generations shows that even with experiments needed to define guiding paths LLGP-GuP needs less time (in number of generations) to generate, with specific probability, strategies with 100% of hits then LLGP.

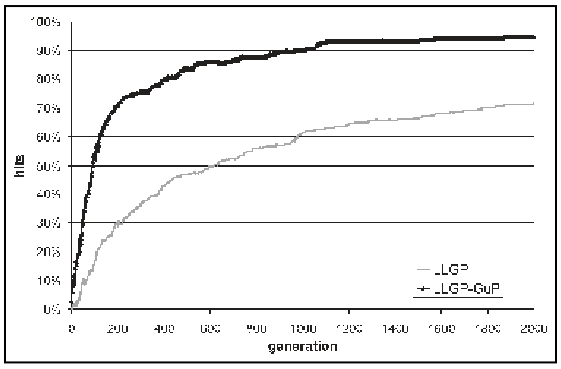

Figure 5.2. An average of experiments best strategies' percentage of hits for succeeding generations.

6 Conclusion and Future Work

In the paper we have experimentally shown that the Guide-Paths technique shortens time needed to create strategies with layered learning genetic programming and improves their effectiveness. Future work should include more experiments with different values of operators trust coefficients. Methods for rating quality of guiding paths should be defined.

References

1. Aaron E., Goldenstein S., Goswami A., Guibas L., Karavelas M., Metaxas. D., (2001). Scalable nonlinear dynamical systems for agent steering and crowd simulation. In: *Computer and Graphics*, 25(6), 983-998.
2. Bajurnow A., (2004). Layered learning for evolving goal scoring behavior in soccer players, *Proc. of Congress on Evolutionary Computation*, 1828-1835.
3. Blumberg B., Berlin M., Downie M., Ivanov Y., Johnson M. P., Tomlinson. B., (2002). Integrated learning for interactive synthetic characters, *Conference on Computer Graphics and Interactive Techniques*, 417-426.
4. Duthen Y., Luga H., Panatier C., Sanza C., (1999). Adaptive behavior for cooperation: a virtual reality application, *Proc. of International Workshop on Robot and Human Interaction*, 76-81.
5. Gustafson S.M., Hsu W.H., (2002). Genetic programming and multi-agent layered learning by reinforcements, *Proc. of Genetic and Evolutionary Computation - GECCO*, 764-771.
6. Jones R.M., Laird J.E., Nielsen P.E., (1998). Real-time intelligent characters for a non-visual simulation environment. *Proc. of Computer Animation Conference*.
7. Kallmann M., Sevin de E., Thalmann D., (2001). Towards real time virtual human life simulations. In: *Computer Graphics International*, 31-37.
8. Lach E., (2005). Automatic generation of animation of virtual humans, *The VII International Conference on Artificial Intelligence*, 109-115.
9. Lach E., (2006). *Wybrane techniki wspomagające animację autonomicznych postaci wirtualnych*, PhD. Thesis, Gliwice, Poland, Silesian University of Technology.
10. Mittmann M., Remagnino P., Szarowicz A., Francik J., (2003). Automatic acquisition of actions for animated agents, *Proc. of GAME-ON Conference on Simulation and AI in Computer Games*.
11. Reynolds C.W., (1987). Flocks, herds, and schools: A distributed behavioral model. In: *Computer Graphics*, 25-34.
12. Sprinkhuizen-Kuyper I., Spronck P., Ponsen M., Postma E., (2006). Adaptive game AI with dynamic scripting. In: *Machine Learning*, 63(3), 217-248.
13. Tang W., Wan T.R., (2002). Intelligent self –learning characters for computer games, *Proc. of Eurographics UK Conference*, 51-58.
14. Thalmann D., Ulicny B., (2002). Crowd simulation for interactive virtual environments and VR training systems, *Proc. of Eurographic workshop on Computer animation and simulation*, 163-170.
15. Tu X., Terzopoulos D., (1994). Artificial fishes: Physics, locomotion, perception, behavior. *Proc. of SIGGRAPH*, 28, 43-50.
16. Whiteson S., Kohl N., Miikkulainen R., Stone P., (2005). Evolving soccer keepaway players through task decomposition. In: *Machine Learning*, 5-30.
17. Koza J.R., (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press.