Neuro-evolution in Zero-Sum Perfect Information Games on the Android OS

Gabriel Iuhasz and Viorel Negru

Abstract. In recent years significant work has been done to use Neural Networks in game AI, and harness the advantages of such a technique. This paper would like to show that it is possible using neuroevolution to evolve a neural network topology optimized for a given task and avoiding over or under complexification by human hands. In order to illustrate this we implement two agents capable of playing simple zero sum perfect information games with the help of the genetic algorithm Neuroevolution of augmenting topologies. To illustrate this optimization we load the resulting topologies onto an Android OS game app.

AMS Subject Classification (2000). 68T05 ; 68T42; 68N25

Keywords. neural networks; neuroevolution; mobile platform; Game AI; zero sum games; Android OS;

1 Introduction

Artificial Intelligence research has always been particularly attracted to the study of games as a measure of intelligence on the one hand and the connection between human learning and games. Also the fact that they are ubiquitous in modern life computer games are an excellent testbed for artificial intelligence techniques and machine learning [1].

Modern video games have only in recent years begun to explore novel machine learning techniques. This has been done both to increase the longevity of video games and to cut production costs. However there still exists a substantial gap between academic AI and game AI techniques. The most common techniques used in game AI are: Finite State Machines, Fuzzy Logic and Scripting. These techniques are relatively old and/or lack flexibility because a human has to program each state and the transition between them. In addition, video games have gained popularity as common scenarios for implementing (AI) techniques and testing cooperative behavior [2]. The player, an agent, in video games performs actions in a real time stochastic environment, where units controlled by the human have to fulfill a series of finite and clear goals to win the game. Such properties make games an ideal environment to compare machine learning techniques. One shining example is the Black & White series in which the player can train through reinforcement learning a series of animals that use simplified perceptrons to represent the creatures desire and goals [3].

In modern video games there is very little adaptation. Game agents are unable to adapt to new game scenarios: they always react the same way for the same in game situation, thus the game becomes predictable and boring fairly quickly. One way to solve this problem is to use machine learning techniques to train the game agent behavior. The majority of learning techniques have two problems: because the correct behavior is not always known game agents become unpredictable thus ruining the game experience; secondly most techniques do not train the agents during gameplay making these too slow to be considered a viable solution. In this paper we use an adaptation of Neuro-Evolution Through Augmenting Topologies NEAT [4] called real-time NEAT (rtNEAT). As its name implies this algorithm is capable of learning ever more complex behaviors during gameplay and by doing this it also reduces unpredictability by eliminating from the game agent population the least competitive individuals early.

A successful implementation of NEAT in games was used to improve the cooperation between the ghosts to catch a Pac-man human player in the game. Van der Heijden [5], implemented opponent modeling algorithms (via machine learning techniques) to select dynamic formations in real-time strategic games. However, the game industry is reluctant to use machine learning techniques, because knowledge acquisition and entity coordination with current methods is expensive in time and resources. This led to the use of non-adaptive techniques. A major disadvantage of the non-adaptive approach is that once a weakness is discovered, nothing stops the human player from exploiting that weakness. In addition game agents are self interested and because of this it is difficult to make them cooperate and achieve

a coordinated group behavior. A possible solution to this problem is to train the game AI online (i.e. during gameplay) thus being able to dynamically adapt to the human players style.

A mobile game is a video game that is played on a mobile platform such as smartphones, PDA's, portable media players etc. As the number of these devices increased substantially in the last few years so did the desire of consumers for ever more computing power from these devices. With the increase in computing power came the need for ever more complex games. As such computer games of 4-5 years ago began to appear on mobile platforms (ex. Quake III, Starcraft were ported to many Android Phones) [6].

In this paper we will illustrate the capability of the NEAT, proposed by Kenneth O. Stanley in 2004, to optimize neural network topologies for a given task. First we will detail the methodology of implementing a set of simple perfect information zero sum playing agents, the games being Tic Tac Toe and Connect 5. After that we will show that the traditional neural network playing agent is outperformed by the NEAT genetic algorithm trained one by means of empirical testing. To further prove this point we load the evolved topology into a simple Android app, NeatPlay to show the benefits of evolving topologies instead of designing one [4].

We show that by eliminating the human element from designing neural network topologies we can use neural networks on mobile platforms instead of more main stream artificial intelligence techniques. A well designed neural network can perform as well as if not better than these techniques. Furthermore we want to show that from the plethora of neural network training and evolutionary techniques NEAT is well suited for this task.

2 Previous Work

GAR (Galactic Arms Race) is both a multiplayer online video game and an experiment in automatic content generation driven by player preferences. Unique game content, namely spaceship weapon systems, automatically evolves based on player behavior through a specialized algorithm called cgNEAT (content-generating NEAT). In particular, new variants of weapons that players like are continually created by the game itself.

The first application of rtNEAT is a video game called Neuro-Evolving Robotic Operatives, or NERO. The game starts with the deployment of individual robots in a sandbox and train them to some desired tactical behavior. Once the training is complete the game allows the trained robots to be pitted against another trained robot army to see which training regiment is better

[7].

David B. Fogel created a checkers playing program called Blondie24 which was able to reach a high level of play by using an evolutionary algorithm. It was based on a minimax algorithm game tree in which the evaluation function was an artificial neural network. This neural network received a vector representing the current board positions and returned a single value which is passed on the minimax algorithm. An evolutionary algorithm was used to evolve the networks connection weights. This evolution was made possible by competitive coevolution, meaning two competing populations played games against each other receiving +1 points for a win, 0 for a draw, and -2 for a loss. This resulted in an evolutionary process designed to weed out ineffective checkers players. Blondie24 is significant because the game playing expertise is not based on human expertise but rather generated by the evolutionary process itself [2].

3 Methodology

There are 2 zero sum playing agents tested in this paper. The first one is called FFANNPlayer, which stands for feed-forward artificial neural network player. This player will use as its evaluation function a feed-forward neural network. Its inputs will consist out of the current board state (in the case of tic-tac-toe 9 inputs), a fully connected hidden layer and an output neuron which can take the values from $[-1,1]$, the closer the value is to 1 the better the board position. The connection weights are evolved using a genetic algorithm that adjust these in accordance with the evaluation function. The neural network weights are represented as chromosomes. The best chromosomes are determined and mate using crossover.

The second agent is called NeatPlayer. There are several key differences between the two neural networks used in these agents. The first difference is that instead of just adjusting the weights using a generic genetic algorithm the networks topology will also suffer changes using NEAT. NEAT is a genetic algorithm whose genome includes a list of connection genes, each of which refers to two node genes being connected. The connection genes specify the in-node, out-node, the weight of the connection, whether the connection is enabled, and an innovation number, which allows finding corresponding genes during crossover.

The second difference is that historical markings are used as a way of identifying homology by matching up genes with the same innovation number (i.e. same historical origin). Thus NEAT keeps track of the historical

origin for every gene in the system. Speciation accomplishes the protection of innovation, and the small size of evolved networks is ensured by starting out with a population of networks with no hidden nodes.

The NEAT genetic algorithm uses incremental complexification namely that in the first generation the network starts with a minimal topology (fully connected input and output nodes). New neurons and inter neuron connections are incrementally added by mutation operators through the course of the evolutionary process of the networks topology.

It also uses genomes with history tracking, the network's topology is encoded in linear genome. Genes representing neurons and connections have associated so called innovation numbers. When a new neuron or connection gene is introduced, it receives a global innovation number by one higher than the last added gene. Genes with the same innovation number origin from the same common ancestor thus they will likely serve a similar function in both parents. NEAT's crossover operator exploits this observation, genes with matching innovation numbers are randomly chosen from both parents for the new offspring, the rest of genes are taken from the more fit parent.

Speciation and fitness sharing is used. Through speciation the individuals in a generation are arranged into non-overlapping species based on the topology they have evolved and the historical markers in their genome. Individuals mate only with the same species "increasing the a chance that the crossover will produce meaningful offspring. The number of offspring is proportional to the summed fitness of the species" this protects more complex networks that have usually lower fitness at the beginning thus protecting potentially meaningful additions to the genepool. In other words members of a species compete only within their species.

Two simple zero-sum perfect information games were used in the experiment, namely Tic-tac-toe and Connect 5. The first game, although a very simple one, can be used as a proof of concept and in fact it has been used extensively in neural network research to evaluate new neural network techniques. Tic-Tac-Toe is played on a 3x3 board. The objective of this game is to place three markers in a row be it horizontally or vertically. Once these markers are placed the player wins whilst it's opponent losses. Failing a win, a draw can still be forced by the player by preventing it's opponent from placing three markers in a row. The game is sufficiently complex to demonstrate the potential for evolving neural networks as strategies, with particular attention given to devising suitable tactics without utilizing expert knowledge. That is, rather than relying on traditional artificial intelligence techniques which are commonly designed or inspired by a human expert, neural network players can evolve their own strategy based only on there win, loss, draw records.

After the proof of concept has been done on Tic-Tac-Toe, a more complex game, Connect 5, has been chosen to scale up. We used a random player, which makes its moves without taking the last move of its opponent into account, and MINIMAX player, which uses minimax with alpha-beta pruning to calculate the best possible move for the current board position. In Connect-5, simple rules lead to a highly complex game, played on the 49 intersections of 7 horizontal and 7 vertical lines. Two players, X and O, move in turn by placing a stone of their own symbol on an empty intersection. X starts the game. The player who first makes a line of five consecutive symbols of the same type wins the game. Its state space complexity is 10^{172} and its game tree complexity is 10^{70} on a board of 15 x 15. A 7x7 board is used for convenience because of the time it would take a MINIMAX player to train the neural players. Also when playing the game on a smartphone or similar sized mobile device a larger board would be impractical.

In the first stage the agents will be trained by hand coded opponents. This is infact reinforcement learning, in which we only work with observed board patterns and no a priori notion of the games rules. In most cases this type of learning uses training data which is usually all the states in which a board game can exist and the desired output of the network for those board configuration. This approach however is not practical because in games with a large state space these can be exceptionally large. However genetic programming and neural networks can be a viable solution to this problem. The hand coded opponent will play against a neural network thus training it. The Neural Network consists in the beginning of 10 input neurons and one output neuron. The input represents the game board states, in the case of Tic-Tac-Toe it is made of 9 squares, and the bias value. The output represents the value of the board state the neural network assigns to it (from -1 to 1) namely if the proposed board position is favorable to the agent. Because in most board games the player who makes the first move has the advantage 2 matches per population member will be performed each player taking turn to make the first move [9].

After the evolution phase is completed a champion chromosome is identified for each generation. The number of species obtained through speciation, scores and sizes are displayed. The best performing phenotype is isolated and the network topology is expressed by the number of hidden nodes and the number of connections between these nodes. This champion chromosomes was loaded into the NeuralBoardGames game app on the Android OS. By being able to play a challenging game against a neural network on such a limiting mobile platform will show the degree of optimization the neural network topology suffered by evolving it using NEAT.

This technique can be used to train neural networks for any number of

zero sum games, they could also be trained by a human player but this is not feasible because in contrast to a hand coded computer opponent a human player can play the game at a much slower rate and is error-prone. During the training phase the neural network agents and the opponent algorithm take turns in who makes the first move because the player who makes the first move has the advantage as demonstrated by L.V. Allis, H.J. Van den Herik and M.P. Huntjens [10] with their thread-space search and proof-number search techniques used in the "Victoria" program.

During the course of the experiment it has to be noted that if the neural network trains with a perfect playing MINIMAX algorithm the training takes longer and the result is far from optimal. It can take many hours to train a viable neural network that can play tic-tac-toe proficiently. During testing it was noted that training with the random player yielded a faster initial convergence towards an acceptable fitness value. When the ANN can beat the random player a more advanced player is selected for training. This player escalation should be done until we reach the best player which in the case of tic-tac-toe is the MINIMAX player with alpha beta pruning.

As stated above the Neural Board Games app will be capable of loading evolved neural network topologies and was programmed to run under version 1.6 of the Android OS. Once loaded these can be used as a bootstrap and further train the NeatPlayer either by hand or against the hard coded players also present in the app. A new neural network player can be created and trained directly using this app. However the option of loading a preevolved topology was chosen in order to save time and because of some hardware limitations [11].

The app uses a modified version of a NEAT implementation. The modifications were necessary because of the customized version of the Java programming language used by the OS. The main difficulty arose because of the conversion of Java classes into a Dalvik executable. Dalvik being a specialized virtual machine [12].

4 Results

4.1 Tic Tac Toe

The NEAT networks have been trained for 100 generations and a starting population of 250. The activation function is a standard sigmoid function and a linear function for the inputs. The three training sessions ran for about 8 minutes each and produced 3 champions. The best NeatPlayer has a fitness

Figure 1: The left image represents the starting topology and the right image represents the champion topology

score of 0.815 of a theoretical max of 1. The starting topology consisted of the input neurons and an output neuron which are fully connected. See Figure 1.

From the starting topology NEAT evolved the following topology for the NeatPlayer champion; 2 hidden nodes and 105 evolved connections in the case of Tic-Tac-Toe. This relatively high number of connections can be explained through the capability of NEAT to evolve recurrent connections and the capability to disable and re-enable evolved connections. This champion comes from the second NeatPlayer training session and it evolved from a number of 50 species. Figure 2 represent a graphical representation of fitness progression during training, the blue lines represent the fitness on the y axis.

Figure 2: The number of generations is displayed along the Ox -axis (100) while fitness along the Oy axis labeled in dark green; the black dots represent the average fitness for each generation

Figure 3: The number of genes are represented in the Oy axis the blue lines represent the minimum and maximum number of genes for each generation, the black dots represent the average complexity and the red triangles denote the complexity of the fittest individual (neurons and connections)

As shown in Fig.3 the NEAT network evolves its topology thus it can find a higher fitness score with fewer neurons than one designed by a human.

Avg.Neat	vs.Random	vs.MiniMax
win	59.90%	0.00%
lost	27.13%	55.17%
tie	12.97%	44.83%
Neat Champ	vs.Random	vs.MiniMax
win	65.00%	0.00%
lost	13.00%	17.00%
tie	22.00%	83.00%

Table 1: (top) The average result of the three training sessions of the Neat-Player; (down) The result of the NeatPlayer champion versus the hard-coded players

Table 1 (top) represents the average win loss and tie versus the two hand coded opponents during the 10.000 matches played during training. Table 2 (bottom) represents the same win, loss, tie rate for the NeatPlayer champion from the second training session.

It should be noted that the NeatPlayer champion is in no way a perfect player. Our goal was not to evolve a perfect player but to show that it is possible for a relatively simple neural network outperform a much more

complex one provided it is designed specifically for that task. Theoretically eliminating over complexification.

The FFANNPlayer was trained with the help of a genetic algorithm similar to the NeatPlayer, with the notable exception that it has a fixed topology. The genetic algorithm was used only to evolve the network connection weights. It has 9 input neurons plus the bias, 15 hidden neurons and one output neuron. We wish to compare the performance of the two neural network playing agents to show the optimizing potential of NEAT. The same MINIMAX and random player was used to train it. The number of hidden neurons are the same as used in [2] by David B. Fogel. The training took approximately 1 minute less than in the case of the NeatPlayer but this fact can be explained by the additional calculations needed by the processor to add new topology in the case of NEAT. The champion FFANNPlayer has a fitness score of 0.78 out of the maximum of 1.

Avg.FFANN	vs.Random	vs.MiniMax
win	52.02%	0.00%
lost	33.82%	68.92%
tie	12.89%	31.08%
FFANN Champ	vs.Random	vs.MiniMax
win	54.00%	0.00%
lost	30.00%	17.00%
tie	16.00%	83.00%

Table 2: (top) Average results during training of the FFANN versus the hand-coded players; (bottom) The results of the FFANN Champion versus the hand-coded players

By comparing the training result of the FFANN player (Figure 4) with that of the NEAT player it is obvious that the later statistically did a better job in learning to play Tic Tac Toe although it had a smaller number of neurons but it had a different topology not all hidden neurons being on a single layer as is the case with the FFANNPlayer.

It can be seen that statistically the NeatPlayer is better at learning to play Tic Tac Toe than the FFANNPlayer. Though in the course of 100 generations the players that have evolved didn't reach a perfect fitness score. The similarity of the two champions during their matches with the hand-coded opponents is a direct result of two things: first the players don't have a perfect score thus their games against the random player will suffer significant statistical fluctuations, making it difficult to accurately gauge which one is in fact better; second a neural network needs approximately 48 hidden nodes

Figure 4: Fitness evolution for the standard FFANN trained with generic algorithm

to remember all the winning combinations during training, this has been explored in detail by Colin Fahey [15]. In fact the weight system in a neural network is its memory. Thus the number of nodes a FFANN has governs its capability of learning complex tasks. Because NEAT evolves its topology it is capable of having less nodes but more connections between them than a FFANN. It also should be noted that the distribution of hidden neurons into hidden layers is also important. After 100 matches between the NeatPlayer and the FFANNPlayer the former won by 52% of the time. The NeatPlayer champion chromosome was loaded on to the NeuralBoardGames application for the Android OS. This migration proved that the evolved topology was well suited for such a limiting hardware platform.

4.2 Connect 5

In the case of Connect 5 the setup remained largely unchanged. One difference is the starting neural network topology will consist in 7 input nodes, 40 hidden nodes and 7 output neurons. The presence of pre-evolution hidden nodes is explained by the fact that if these were not present a much larger number of generations would be required to accurately gauge the performance of the NeatPlayer agent. One of the hardcoded players that will train the neural network is also changed because training with a MINIMAX with alpha-beta pruning would require an extremely long time. A defensive player was used that seeks to block the neural network players winning moves this player is called the LogicPlayer.

Even so we have noticed that the time necessary to play 10.000 games is extremely long thus it has been shortened to 3500 games. Some runs of 100+ generations were made but no real difference in fitness score has been observed.

Avg.Neat	vs.Random	vs.Logic	Neat Champ	vs.Random	vs.Logic
win	76.35%	32.54%	win	94.00%	75.00%
lost	18.53%	60.91%	lost	4.00%	24.00%
tie	3.20%	5.60%	tie	2.00%	1.00%

Table 3: (top) The average results of training sessions of NeatPlayer; (bottom) How the NeatPlayer champion did against the hard-coded players

In the case of 100 generations about 4 hours are needed for training, and only when reaching 800+ generations is the difference significant, but because of the training time it is extremely impractical in the case of this paper. The champion NeatPlayer has a fitness score of 0.87 from a maximum possible of 1, 139 hidden nodes and 4901 connections (Figure 5). It should be noted that because none of the hard-coded players play a perfect game even if the champion would reach the maximum fitness score (Figure 6).

Figure 5: x-axis represents the number of generations while the y -axis represents fitness ; the black dots represent the average fitness for each generations

The only difference in the case of Connect 5 is that this player uses a fixed starting topology, with 40 hidden fully connected neurons. Also a slightly lower training time has been observed, it took 2 minutes less than in the case of the NeatPlayer.

Figure 6: The complexity of the evolved topologies during training; it can be seen here that the fittest individual in the population is not always the one with the greatest number of neurons

As seen before the statistical difference between the two players is significant, the NeatPlayer being better. During the versus matches the latter won consistently 70% of the games. This lead to the conclusion that the NeatPlayer is better at learning relatively complicated games such as Connect-5 than a fixed topology network. The FFANNPlayer's starting topology proving to be too primitive. A MINIMAX player was also implemented with training in mind but at the relatively small search depth of 2 it took 2-3 times as long to train the neural networks. On the Andoird OS the same MINIMAX player highlighted the fact that the NeatPlayer can pick the best move much faster than the former can.

It should be noted that the identical starting topology for both of the players meant to show how a particular fixed topology may not be enough to solve a given problem, needing further neurons and/or connections. A neuroevolutionary technique such as NEAT solves this problem.

This approach of scalling the experiment is similar to that in [13][14] where a multidimensional recurrent neural network was used to scale from a small Go board of 5 x 5 to 9 x 9 board.

Avg.FFANN	vs.Random	vs.Logic
win	74.35%	30.28%
lost	18.53%	61.87%
tie	7.12%	7.84%
FFANN Champ	vs.Random	vs.Logic
win	90.00%	72.00%
lost	9.00%	25.00%
tie	1.00%	3.00%

Table 4: (top) Average results of the training sessions of the FFANNPlayer; (bottom) Champion against hard-coded player

5 Conclusions

This paper proposed to compare two types of neural networks and to see which one is best suited for mobile platforms. It has been made clear during the studies that it is well suited for that task. As this non reinforcement type of training is more akin to an art form. Designing a network topology by hand is extremely complicated because of the black box nature of neural networks. During the Tic Tac Toe study it was shown that although the FFANN had more complex topology this was not utilized optimally as the hidden neurons were all put in a single hidden layer whilst the NEAT neural network evolved a topology that was able to get a higher fitness score despite having a smaller number of neurons but a high number of inter neuron connections. In the case of Tic Tac Toe the evolved topology can be considered as an optimized one for the given task given that it outperforms the standard topology.

During training no information regarding the objective of the game was given, also no values to various board patterns. The only measure of progress the neural network were given was the number of won, lost and tied games. Even so the neural networks did a good job in learning to play the game. A similar study was made by David B. Fogel [2] with similar result, being able to produce nearly perfect playing neural network players over 900 generations.

Even after scaling up to Connect 5 NEAT did fairly well. By starting evolution on a preexisting topology of 40 hidden nodes it has been shown that in the course of 35 generations NEAT could reach a better fitness score by complexification. Thus proving that the starting topology was too simple to successfully learn Connect-5. A significant drawback NEAT is the relatively large computational power needed to train it.

The problem of over or under complexification is also addressed in this paper and we manage to highlight that NEAT is able to solve this problem

or at least minimize it to a point where it is no longer a serious problem.

When migrating the evolved neural networks to the Android OS it also has been noted that with an optimized topology a neural player can calculate a move much faster than a player that uses MINIMAX to calculate its next best move. Many current games on the Android OS have to deal with limited computational power and even more limited memory. The average speed of a smartphone CPU is well under the 1 GHz mark and has less than 512 MB of RAM. Because of these limitations many games use outdated or suboptimal artificial intelligence techniques. One example is that of smartphone chess games which cannot hope to reach a low enough game tree level as their PC counterparts. As we saw in the case of our "Neural Board Games" app the neural network evaluation function is a much faster and computationally less expensive option. We can see from the example of Fogels Blondi24 program that a well-trained neural network can perform as well if not better than a traditional evaluation function.

Also while training the neural network players with the random player a much better training rate has been obtained. This is because when playing against a perfect player the neural network can't observe winning board patterns as easily. The game often does not last long enough to teach the neural network new board configurations. In the case of Tic-Tac-Toe this is not immediately obvious because of the limited possible board configurations. These being mirror images of each other. However when scaling up to Connect 5 we have a much bigger board and a much larger number of possible board positions. So if the game lasts only 5 moves the neural network player has to play a much larger number of games in order to learn as many favorable board configurations as possible.

6 Future Work

Because in Tic Tac Toe and Connect 5 there is a limited number of patterns, a rotating neural network could be implemented. This type of network would be able to play a competitive game while having a relatively simple topology. For example in Tic Tac Toe there are only 3 unique starting positions, the rest being only mirror images of these. So a neural network has to learn 3 positions and rotate its input in order to compensate. This is similar to remarks in [10].

A scalable neural network could also be used to play on a small board and then study how the same network plays on a bigger board. In essence a smaller instance of a problem could be used to solve a bigger instance of

the same problem. A similar method is being explored by T. Schaul and J. Schmidhuber [16].

A co-evolution strategy could also be used to train neural networks. By evolving two populations simultaneously and pitting them against each other in a tournament. This approach however creates the need for a different kind of fitness evaluation method. One method proposed for this kind of evolution is "Competitive fitness sharing" which rewards individuals who beat opponents few others have not the ones that beat the most.

Furthermore as a real-time extension to NEAT already exists this type of neuroevolution can be used on more modern games. For NEAT or more precisely rtNEAT can be used on real-time strategy games. Micromanaging of units in these games is particularly difficult because of the sheer number of such units present at any one time in the game environment. These units can be considered agents because they function in an environment and are autonomous in this environment. Traditional artificial intelligence techniques can be extremely costly in terms of memory and computational resources but by evolving a neural network capable of handling extremely low level tasks (i.e. when to retreat/attack/ flank, whom to target etc.)[17][18][19].

Acknowledgement

This work was partially supported by the strategic grant POSDRU/CPP107 /DMI1.5/S/78421, Project ID 78421 (2010), co-financed by the European Social Fund Investing in People, within the Sectoral Operational Programme Human Resources Development 2007–2013.

References

- [1] **A. Narayek**, Intelligent Agents for Computer Games, *Computers and Games Second International Conference*, (2000), 414–422
- [2] **C. Kumar and D. Fogel**, Evolution, neural networks, games, and Intelligence, *Proceedings of the IEEE volume 87 Issue 9*, (2000), 1471-1496.
- [3] **M. Buckland**, AI Techniques For Game Programming, *Premier Press*, (2006), 480.
- [4] **K. Stanley**, *Efficient Evolution of Neural Networks through Complexification*, Department of Computer Science University of Texas, 2004.
- [5] **M. van der Heijden, S. Bakkes, and P. Spronck**, Dynamic formations in real-time strategy games, *Proceeding of the IEEE Symposium on Computational Intelligence*, (2008), 47-54.

- [6] **J. DiMarzio**, Android A Programmers Guide, *McGraw-Hill Osborne Media first edition*, (2008), 400.
- [7] **A. Agogino, L. Stanely, and R. Miikkulainen**, Online Interactive Neuro-evolution, *Neural Processing Letters*, (1999), 100-109.
- [8] **D. Fogel**, *Mind Games - Evolving a Checkers Player Without Relying on Human Experties*, ACM - Intelligence New Visions of AI in Practice, **11, 2**, 2000.
- [9] **J. Schrum**, *Neuro-Evolution in Multi-Player Pente*, <http://www.cs.utexas.edu/~tilde/schrum2/>, 2006.
- [10] **L. V. Allis, H. J. van den Herik, and M. P. H. Huntjens**, *Go-Moku and Threat-Space Search*, Departement of Computer Science University of Limburg, 1993.
- [11] **Jerome J.F. DiMarzino**, Android A Programmers Guid, *McGraw-Hill Osborne Media, First Edition*, (July 2008), 400.
- [12] **Vladimir Silva**, Pro Android Games, *Apress, First Edition*, (2008), 300.
- [13] **Tom Schaul and Jurgen Schmidhuber**, *A Scalable Neural Network Architecture for Board Games*, 2007.
- [14] **M. Gruttner, F. Sehnke, T. Schaul, and J. Schmidhuber**, *Multi-Dimensional Deep Memory Atari-Go Players for Parameter Exploring Policy Gradients*, 2008.
- [15] **C. Fahey**, *Neural network with learning by backward error propagation*, www.colinfahey.com, 2004.
- [16] **T. Schaul and J. Schmidhuber**, A Scalable Neural Network Architecture for Board Games, *Proceedings of the IEEE Symposium on Computational Intelligence in Games*, (2008), 324-334.
- [17] **M. Buro**, Real-time strategy games: A new AI research challenge, *Proceedings of the International Joint Conference on AI*, (2003), 485-486
- [18] **K. O. Stanley, B. D. Bryant, I. Karpov, and R. Miikkulainen**, Real-time evolution of neural networks in the NERO video game, *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, (2008), 302-314.
- [19] **A. Shantia, E. Begue, and M. Wiering**, Connectionist Reinforcement Learning for Intelligent Unit Micro Management in StarCraft, *International Joint Conference on Neural Networks*, (2011), 234-246.

Gabriel Iuhasz and Viorel Negru
Department of Computer Science
West University of Timisoara
Blvd. V. Parvan, Nr. 4
300223 Timisoara
Romania
E-mail: {iuhasz.gabriel,vnegru}@info.uvt.ro

Received: 05.01.2012

Accepted: 11.06.2012