Real-time challenge balance in an RTS game using rtNEAT

Jacob Kaae Olesen, Georgios N. Yannakakis, *Member, IEEE*, and John Hallam

***Abstract*— This paper explores using the NEAT and rtNEAT neuro-evolution methodologies to generate intelligent opponents in real-time strategy (RTS) games. The main objective is to adapt the challenge generated by the game opponents to match the skill of a player in real-time, ultimately leading to a higher entertainment value perceived by a human player of the game. Results indicate the effectiveness of NEAT and rtNEAT but demonstrate their limitations for use in real-time strategy games.**

I. Introduction

Computer game balance (or difficulty adjustment) is a crucial aspect of commercial game development. Currently, this is achieved either by predefined levels of game challenge — the player then decides which of those levels she will play against — or by techniques known as rubber-band artificial intelligence (AI) [1], mainly used in racing games. The first approach cannot incorporate the needs of all potential players of the game while the latter approach generates predictable behaviors which reduce the believability of the non-player characters (NPCs). Furthermore, human players enhance their skills while playing a game which necessitates an adaptive mechanism that covers the player's need for more challenging NPCs during play (i.e. in real-time).

The work presented here is motivated by the current lack of intelligent automated processes that adjust game challenge according to individual player skills: an attempt dynamically to adjust challenge generated by game opponents of real-time strategy (RTS) games in real-time is introduced in this paper. The first step is to identify factors that contribute to the challenge experienced by a player in the RTS game under investigation. Then, the Neuro-Evolution of Augmenting Topologies (NEAT) [2] methodology is used to train off-line artificial neural network (ANN) controlled agents that excel in some of these factors, and the generated behavior is tested to verify that it performs well against challenging opponents. Next, an empirical challenge rating formula is designed based on quantifications of all challenge factors. This challenge metric is then used as a component of a fitness function which promotes minimization of the difference between the challenge metric value generated by the NEAT agent and the one generated by its opponent. The ability of NEAT to balance the challenge metric (i.e. minimizing the above-mentioned difference) is first tested off-line. Results show that NEAT is capable of matching the challenge of the AI agent to the skill of a hard-coded player throughout the whole evaluation gameplay period. Based on these positive indications, the real-time NEAT (rtNEAT) methodology [3] is used to adjust the challenge of ANN-controlled NPCs according to the player's challenge in real-time. Experimental results indicate the efficiency of rtNEAT in the challenge-balance task for the RTS game investigated and demonstrate the limitations of the approach.

The work reported here is novel in demonstrating a way of constructing a challenge metric for the RTS game used and in applying NEAT and rtNEAT for dynamically adjusting challenge in RTS games. The limitations of the proposed methodology and its generic use as an efficient approach for automating game balance in RTS games in real-time are discussed.

Jacob Kaae Olesen and John Hallam are with the Mærsk Mc-Kinney Møller Institute, University of Southern Denmark, Campusvej 55, DK-5230, Odense (email: joles03@student.sdu.dk, john@mmmi.sdu.dk). Georgios N. Yannakakis is with Center for Computer Games Research, IT-University of Copenhagen, Rued Langgaards Vej 7, DK-2300 Copenhagen (phone: +45-7218-5078; fax: +45-7218-5001; email: yannakakis@itu.dk).

II. Related Work

Theoretical psychological studies based on computer game players suggest that the appropriate level of challenge is an important factor of an enjoyable game experience. Malone's analysis on principal entertainment factors [4], suggests *challenge* as one of the three intrinsic qualitative factors that contribute to engaging game play (*curiosity* and *fantasy* are the other two). Challenge's contribution to enjoyable game-play experience is also derived from the well-know *theory of flow* [5], since according to Csikszentmihalyi appropriate level of challenge that matches player skill constitutes one of the nine factors of *flow* generated during play [6]. Lazzaro's work on 'fun' clustering [7] derived from facial expressions and data obtained from game surveys on players reveals *hard fun* as one of the proposed entertainment factors which corresponds to real-time game challenge. Yannakakis and Hallam propose a human cross-verified interest metric for prey/predator games that includes a quantified measure of the appropriate level of challenge [8].

Based on the assumption that challenge is the only factor that contributes to enjoyable gaming experience, several machine learning approaches have been proposed for adjusting a game's difficulty. Such approaches include applications of reinforcement learning [9], genetic algorithms [10], probabilistic models [11] and dynamic scripting [12], [13]. However, human survey experiments that cross-verify the assumptions of player satisfaction enhancement have not been reported in any of those approaches. Using challenge as a player interest factor, an adaptive (neuro-evolution) learning mechanism has been proposed for increasing the challenge-based interest value of prey/predator games. The

978-1-4244-2974-5/08/$25.00 ©2008 IEEE

effectiveness and robustness of the mechanism has been evaluated via human survey experiments [8].

Following the theoretical principles reported in [4], [14] and [8], this paper is primarily focused on the contributions of game opponents' behavior to the real-time entertainment value of the game through game challenge balance. Given the successful application of the NEAT methodology [2] in several complex problems and rtNEAT in evolving adaptive NPC behaviors in real-time for the NERO game [3], we investigate those algorithms' ability to balance challenge in real-time on a test-bed RTS game.

III. NEAT

NEAT is a method developed by Stanley and Miikkulainen [2] used to construct artificial neural networks automatically which is based on Topology and Weight Evolving Artificial Neural Networks (TWEANNs). Evolution adjusts both the connection weights and the topology of the network. This allows NEAT to add and remove nodes and connections in the ANN, as well as modify the values of the connection weights [15]. Common neuro-evolution methods — that use fixed ad-hoc topologies and only allow evolution of connection weights — have previously been much faster than TWEANNs, since the complexity introduced in the evolutionary process by dynamic topology resulted in lower performance. NEAT was developed in an effort to improve TWEANNs' performance and outperform neuro-evolution methods using fixed topologies. It does so by addressing three critical issues which are briefly presented below.

A. Simple and meaningful crossover between different topologies

The apparently simple problem of performing crossover between two ANNs in a population can become very difficult if their topologies are very different. NEAT uses a linear encoding of all network connectivity in each genome, resulting in a list of connection and node genes. Each connection gene includes the codes of the input and output nodes of the connection and its weight value. In addition, it incorporates an enable/disable flag and an innovation number, unique for each gene, which is used to keep track of when this specific connection was introduced in the population. During crossover, the genes (connections) with the same innovation number are lined up against each other, and the rest are either *disjoint* or *excess* genes, depending on whether they are within or outside the range of the other parent's innovation numbers. The generated offspring will randomly inherit matching genes from both parents, but excess or disjoint genes only from the more fit parent.

Mutation in NEAT allows for change of both connection weights and network structure. Structural mutations may happen in two ways: by adding a connection gene between two unconnected nodes or by adding a node gene, splitting an old connection gene into two new, and disabling the old. The reader may refer to [2] for further details on the genetic operators used in NEAT.

B. Protection of structural innovation using speciation

Using NEAT, ANN topology is expanded by mutations adding new connection or node genes to the genome. However, adding complexity to a network structure usually leads to an initial decrease of fitness, until the structure is optimized after a number of generations. This can cause newly generated and potentially fit genomes to be eliminated during the evolutionary process. NEAT overcomes this by speciation which allows the genomes to compete only against similar genomes — i.e. genomes of the same species. In speciation, all genomes are divided into species by clustering using a compatibility distance measure which is calculated as a linear function of the excess and disjoint genes between two genomes. A compatibility distance threshold determines the number of species clusters formed.

C. Minimizing dimensionality

NEAT performs fast because of an incrementally growing ANN structure evolutionary procedure. Where other TWEANNs usually initialize populations of random topologies to gain topological diversity, NEAT starts out with a uniform population of fully connected networks of minimal size (i.e. no hidden nodes). Topology is then expanded and diversified by mutation and speciation, but since the initial topologies are as simple as possible, the search space is structured so that, unlike other TWEANNs and fixed-topology neuro-evolution systems, NEAT only expands on topologies that have already survived evolution and proven to be fit.

NEAT has been successfully used on traditional AI problems, such as the pole-balancing problem [2], [16] in which it is able to surpass the performance of most evolutionary methods and produce innovative ANNs able to solve highly complex problems. A more detailed presentation of NEAT can be found in [2]; unless otherwise stated, NEAT is applied here using the default parameters as presented there. Experiments for customizing specific parameters of NEAT for this application are presented in Section VII. Further details on the application of NEAT and rtNEAT in the *Globulation 2* RTS game can be found in [17].

IV. GLOBULATION 2

Globulation 2 (G2) [18] is an open-source RTS game, implemented in C++ by a large group of independent game developers. The game is currently in a beta stage, but is robust enough for the purposes of this paper. The game is based on the well known concept of resource harvesting and base-building to create an army and defeat the opponent(s) (see Fig. 1).

G2 incorporates some interesting features that set it apart from most other RTS games. Where most RTS games require a large amount of micro-management by the player to control all buildings and units, G2 allows the player to concern herself only with macro-management. This is achieved by letting the units determine their tasks and roles based on high level orders, rather than by player assignment. If, for

Fig. 1. Globulation 2 test-bed game: screenshot during a combat.

example, the player wishes to have three units harvesting food, she simply specifies so in the building where food is needed, and three worker units will automatically begin delivering food to that building. Similarly, constructing new buildings is done just by specifying where the new building should be placed and how many workers should work on this task. Units can be upgraded by certain kinds of building, such as schools and racetracks, and enter those buildings whenever they pass near them. The only decision the player has to make is how many such upgrade buildings should be made, and where to place them.

Finally, combat is also simplified, in that warriors will automatically attack any enemy in their area of coverage. It is also possible to mark an area to be guarded, and warriors will automatically rally to that point when they are not otherwise occupied. When performing an attack, the player simply specifies a circle on the map to be attacked. The radius of the circle can be adjusted, as well as the number of warriors to attack the area. The circle can be made small enough to specify a single building, or to encompass an entire enemy base. Warriors will then move to the attack area and start attacking any enemy within it, usually starting with enemy warriors if any are present.

As with most RTS games, construction in G2 is based on resources. To create buildings, resources such as wood, stone and algae must be collected by workers. Furthermore, creating new units requires food to be harvested. Moreover, G2 also uses food to maintain units. This is done in inns which both workers and warriors must visit at regular intervals to feed. These inns must be supplied with food by workers, and can only contain a limited number of simultaneously feeding units. If no inns are available once a unit becomes hungry, it will starve, eventually lose health and finally die. This adds an extra strategic element to G2, as warriors attacking a position far from an inn spend most of their time running back and forth to the inn in order to feed, and very little time fighting.

The focus on macro-management in G2 makes it an excellent test-bed for AI development, since an AI agent only has to make high level decisions to operate on the same level as a human player. This makes an ANN-based AI much simpler to develop for G2, as the search space (number of available actions) is greatly reduced compared to other RTS games.

V. Challenge Rating

To adjust the challenge of the game to meet the player's skills, the notion of challenge must be quantified first: a metric must be defined for G2, enabling continuous measurement of challenge — both of the AI and the human player. This metric is named the 'challenge rating' (CR) and is inspired by previous studies in the field of entertainment modeling [8] and dynamic game balancing [9].

First, in order to design a reliable CR for G2, it is necessary to examine the specific game features (e.g. macro-management) that differentiate G2 from other RTS games (see Section IV). Based on knowledge of the RTS genre, several forum posts and interviews with a number of experienced computer gamers, the following elements are proposed as the main contributors to challenge in G2.

- Number of warriors
- Aggressiveness
- Infrastructure
- Defense infrastructure
- Average training level of warriors
- Number of workers

(Following the principles of [8] these elements are only based on NPC behavior. Thus, game features like map layout and number of players are excluded from investigation.)

There is no guarantee that these proposed challenge factors do actually contribute to challenge in the G2 game. The contribution of each factor to challenge might also vary widely. To determine the validity of each challenge factor, and possibly also compute their quantitative contribution (e.g. weighting factor) to the challenge metric, a number of AI agents must be trained off-line, each focusing on maximizing one or more challenge factors. A number of games must then be played by human test subjects to cross-validate whether each of the produced AI behaviors matches the human notion of challenge [8]. Once these tests have been completed, a definition of a total challenge rating can be derived by using those challenge factors proven to generate challenging games for humans. The generated CR metric — being an approximator of challenge — must then be cross-validated in the same way as the individual challenge factors. By following the experimental methodology introduced in [8] a number of NPCs must be evolved, reaching different levels of challenge, according to the CR metric.

In the work presented here, off-line learning (via NEAT) against standard AI agents that come with G2, is used to verify whether the first two of the aforementioned challenge factors (number of warriors and aggressiveness) do indeed contribute to challenge (see Section VII). Our assumption

here is that the standard G2 agents offer a plethora of effective playing behaviors which simulate human players and are adequate to demonstrate whether a specific challenge factor contributes to human challenge. Then, a challenge rating formula is designed based upon all those six factors (see Section VII). The ad-hoc formula designed provides a good estimation of the challenge of the game based on empirical tests. Although many alternative and more successful challenge metrics could be designed, the one proposed in Section VII-C serves the purpose of this paper well by providing an objective function which demonstrates the ability of machine learning to adapt the game in real-time.

VI. AI Design

The off-line development of the many different performing NPCs described in Section V is achieved through NEAT in this paper. Other machine learning approaches, such as fixed-topology neuro-evolution or temporal difference learning, could be used. However NEAT is the most sensible approach for off-line learning due to compatibility purposes since it is used for real-time adaptation of the game challenge.

This section presents the ANN structure used, the test-bed game level designed for all learning experiments conducted in G2 and the specifics of the evolutionary procedures of NEAT and rtNEAT.

A. ANN structure

G2 comes with several scripted AI agents developed by enthusiastic players of the game. It was decided to base our AI implementation on an existing AI, rather than create it from scratch, for several reasons. First, it is effort and cost efficient since the methods and structure of the AI are already implemented and tested. Second, a few of the AI agents that come with the game already have a structure which supports well the implementation of a neural network controller, in that they use a number of statistical game data as input to determine which actions to take, every given number of game ticks. These AI approaches use simple hard-coded rules for action selection. It is, therefore, a relatively trivial task to replace this standard rule-base with an ANN, by feeding it with the same input vector. Finally, building upon an existing AI agent also provides a reliable baseline for comparison of the generated behavior with the existing one.

After observing the behavior of each available standard AI agent during game play and going through their rule-base systems in detail, we chose the agent named *Nicowar* to form the baseline NPC for our investigations. The selection criteria include performance, documentation availability and structure compatibility with an ANN. Following the input-output vectors of the *Nicowar* agent, the initial ANN structure incorporates 8 inputs and 8 outputs presented in Table I with no hidden layer. This results to 64 connections, plus 8 additional connections for the output bias.

Input is normalized into $[0, 1]$. Each output corresponds to a macro-management decision that the agent might take. Based on *Nicowar*, these decisions correspond to activating different game phases with associated playing strategy. The input is calculated every 100 game ticks and the output is updated. Then the agent bases its decision on all active output nodes since more than one output can be active simultaneously. Further details on each of those game phases can be found in [18].

TABLE I
ANN Input And Output.

Input	Output
Total number of units	Growth
Number of warriors	Skilled worker
Number of idle workers	Upgrade 1
Number of rank 1 workers	Upgrade 2
Number of rank 2 workers	War preparation
Number of barracks	War
Number of schools	Starvation recovery
Starvation percentage	No workers

B. Game Environment

The map used for all training experiments presented in this paper is a modified version of an existing map named *Muka*. This map is highly symmetrical providing fairness with regards to the NPCs starting position. For our implementations a *Muka* map is designed providing one warrior and one worker development building as well as 9 warriors and 19 workers. This enhanced map allows the AI training a much simpler platform, as it can only focus on the development of units during the game.

C. Evolution through NEAT

The evolutionary procedure used follows the standard NEAT procedure presented in [2]. First, a population of ANN-controlled agents is created. Then, each member of the population is placed in the game environment, evaluated on a specific training task and assigned a corresponding fitness value. Selection is the standard NEAT one. Finally, the population is re-spawned with the probability of mutation and crossover occurring in all agents.

D. Evolution through rtNEAT

Real time learning is based on the principles of rtNEAT implemented for the NERO game [3] where the ANNs that control the opponents are evolved in real-time. In NERO, a large number (50) of agents are used to provide sufficient interaction and the real-time replacement strategy occurs quite often, which results in fast adaptation. In G2, however only 8 agents (players) can be active in a game simultaneously, given the game environment and genre. Additionally, the replacement strategy cannot be that frequent, as it takes quite a while to obtain sufficient statistical data from an RTS game in order to evaluate well the fitness of the generated behavior. These constraints constitute challenges for the successful implementation of rtNEAT to G2. However, rtNEAT has relatively a fair amount of adaptation time available since

RTS games in general generate longer game play periods. For instance, the average length of the G2 game played by the authors lasts more that 20 minutes (roughly 30000 game ticks) on a single level.

VII. Off-Line Learning

This section presents off-line learning experiments for adjusting specific NEAT parameters (see Section VII-A), experiments for generating NPCs that maximize two of the challenge factors mentioned in Section V (see Section VII-B) and experiments for evaluating the efficiency of NEAT in minimizing the difference of a designed challenge metric between the ANN-controlled NPC and the player (see Section VII-C).

A. NEAT parameter value investigation

Initially a number of off-line learning trials were conducted to determine three NEAT parameter values: the compatibility threshold (*ct*), the population size and the evaluation period. These three parameters were chosen for their importance in any evolutionary procedure. NEAT allows for a great amount of parameter adjustment, but testing all of them would be a very tedious task (which, if necessary, would argue against the applicability of NEAT itself). The remaining NEAT parameters were left at their default values [2].

This part of the off-line training procedure was performed using a simple fitness evaluation based on the total number of warrior units at the end of each evaluation period. This fitness function was chosen to test the efficiency of NEAT in simple problems and provide insight into potential problems during training. The NEAT-controlled agents were evaluated against the *Nicowar* agent [18] that comes with G2. The NEAT and the *Nicowar* agents placed in G2 are allied in this set of experiments.

The initial off-line training trials revealed that the learning process was quite fast and efficient within a reasonable range of the parameters under investigation. After several parameter set values were tested it was observed that a population size of 25 with a *ct* value of 3 and a evaluation time of 10000 ticks generated the most effective agents in 10 trials. The main criterion used for defining effectiveness is based on a compromise between the learning time and the quality of the training result. Each one of the 10 trials lasted 7 minutes running in an Intel Core 2 Duo — 2.2 GHz CPU (all experiments presented in this paper run on that same machine). Moreover, during training, each game tick corresponds to around 1*ms*, making an evaluation of 10000 ticks last 10 seconds approximately. Note that the chosen evaluation period of 10000 ticks corresponds to approximately 6.6 minutes of real-time play (a game tick during play in G2 lasts for 40 *ms*) — much less than the gameplay duration of an average skilled human player.

B. Training Experiments — Challenge Factors

After validating the effectiveness of NEAT in evolving simple controllers, NPCs were trained to match the two first challenge factors mentioned in Section V: *number of warriors* and *aggressiveness*.

1) The MaxWar *agent:* The first NEAT training attempt is built on the *Nicowar* standard agent and it is trained to play against it. The purpose of the learning experiments presented here is to determine if it is plausible to train an agent that develops as many warriors as possible and investigate if it is more challenging to play against that agent than an agent which does not embed that behavior. Thus, the fitness function used promotes the number of warriors built and weighs warriors with combat training higher than warriors without it.

The best trained agent, named *Maxwar*, was evaluated against all six standard AI agents and an agent incorporating a random action selection mechanism. The *Maxwar* agent resulted in a draw against all the opponents within a time frame of 25000 ticks. While this does not suggest the *Maxwar* is incredibly hard to play against, it is still promising that it does not lose to any of the standard AI agents — not even *Nicowar* on which it was based. Taking into consideration that *Maxwar* was not explicitly trained to win a game, but to create many and well trained warriors, the results are more than satisfactory at this stage and show that the number of warriors contributes to a challenging game.

2) The WinFast *agent:* Aggressiveness, especially early in the game, has been suggested by nearly all testers of G2 as one of the most important factors of challenge [18]. Thus, the second NEAT training attempt focuses on winning games as fast as possible generating an agent, namely *Winfast*, with a very aggressive behavior. The fitness function that guides the trained agent towards aggressive behavior is $10^4/t_w$ where $t_w \in [0, 20000]$ is the winning time in game ticks — an evaluation period of 20000 is used in this experiment. Moreover, fitness is set to 0 if the agent loses the game and 0.5 if there is a draw.

Experiments demonstrated that an agent with random action-selection loses to *Nicowar* all games in 10 trials within an average of 14500 game ticks. Thus, initializing the machine learning procedure by playing against the *Nicowar* agent would have resulted to very slow training. For this purpose, we follow an incremental learning approach where *Winfast* is trained against the random action-selection agent and the *Nicowar* agent during the first and second phase of the training respectively. We repeat this learning procedure for 10 trials starting from different initial conditions. The best performing *Winfast* agent picked out of the 10 trials wins every game when playing against all standard AI agents that come with G2. The winning times of *Winfast*, in game ticks, can be seen in Table II.

By following the same approach more challenge factors could have been investigated. However, the focus of the paper is to dynamically adjust challenge using real-time learning. To do this some (and not all) challenge factors needed to be validated to guide the design of a challenge metric to be used as an objective function for rtNEAT. But first, machine learning experiments presented in the following subsection,

TABLE II

AVERAGE WINNING TIME OF *Winfast* BY PLAYING AGAINST STANDARD AI AGENTS. AVERAGE VALUES ARE COMPUTED OVER 10 TRIALS.

AI agent	Winning time (ticks)
Nicowar	9601
Random	10000
Warrush	10529
Numbi	12513
ReachToInfinity	13089
Castor	18913

were conducted to demonstrate whether the difference of the challenge metric between the AI agent and the player could be minimized off-line.

C. Training Experiments — Challenge Metric

The aim of the experiments presented in this section is to examine how well an AI agent could be trained using NEAT to match the challenge of its opponent. For this purpose, a challenge rating (CR) metric that combines different challenge factors is designed. A linear correlation between the factors and the CR metric is assumed here. The CR metric is calculated for both the player and the AI agent (opponent) and the fitness function constructed promotes a minimization of the difference between the two CR metrics eventually leading to challenge balance. The chosen CR metric (see eq. (1)) is implemented as a weighted sum of statistical data obtained from the game and embeds all six challenge factors presented in Section V:

$$\mathrm{CR} = \frac{Wa}{a} + \frac{Wo}{b} + \frac{B}{c} + \frac{D}{d} + \frac{A}{e} \qquad (1)$$

W_a is the fitness function used to train the *MaxWar* agent, weighting trained warriors higher than untrained ones and satisfying two out of the six challenge factors presented in Section V: *number* and *average training level of warriors*. W_o is a similar worker rating, giving trained workers more weight than untrained workers, corresponding to the *number of workers* challenge factor. B, D and A are, respectively, the number of buildings, defense towers and attack flags owned by the player. These last three parameters correspond to the *infrastructure*, *defense infrastructure* and *aggressiveness* challenge factors respectively. Finally, a, b, c, d and e are parameter weights with fixed values of 15, 30, 3, 0.5 and 0.5, respectively, which are determined through several gameplay experiments.

The fitness of each AI agent is calculated using eq. (2), where $\mathrm{CR_{AI}}$ and $\mathrm{CR_o}$ are the total challenge ratings of the trained AI and the opponent player respectively. In cases where the difference is very close to 0, the fitness is set to 10^5 to avoid overflow values at infinity.

$$f = \frac{1}{|\mathrm{CR_{AI}} - \mathrm{CR_o}|} \qquad (2)$$

By following the same experimental procedure described in Section VII-A, $1/f$ reaches 0 in 10 generations in most of the 10 learning trials attempted. We believe that the inverse fitness $1/f$ values of the most fit member of the population during the last generation offers more input than the illustration of the fitness value over those 10 generations. Fig. 2 shows the absolute difference between the challenge rating of the AI and the *Nicowar* opponent (inverse fitness value) throughout a 15000 tick game for three training trials. The first 10^4 ticks correspond to the last evaluation of the 10^{th} generation and the 5000 additional ticks are used to evaluate the trained behavior.

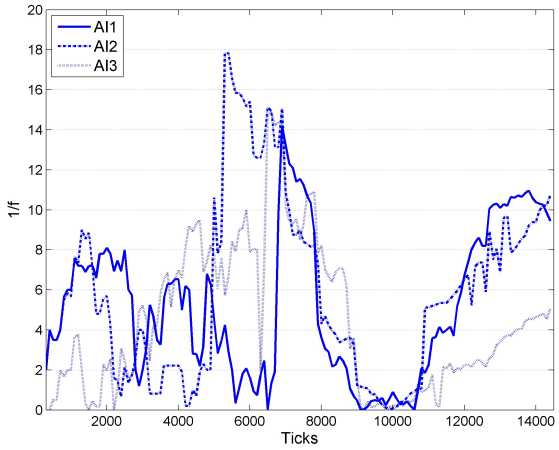

Fig. 2. Fixed evaluation period; Inverse fitness value ($|\mathrm{CR_{AI}} - \mathrm{CR_o}|$) of three most fit agents during the last evaluation of the training procedure.

It can be clearly observed (see Fig. 2) that the AI agents learned the task of minimizing the CR metric difference after 10^4 ticks (the evaluation point) and shows that NEAT can balance the challenge of the AI agent with that of the opponent at a specific time point. It is, however, obvious that the CR difference is not kept low during the whole evaluation period. This expected behavior is not entirely desired, since low $1/f$ values are required during the whole game play period. The testing phase of the experiment shows the inability of NEAT to generalize when learning is turned off.

The main training problem, which is evident from the previous experiment, is that NEAT aims for minimal CR difference only at the end of the evaluation period of 10^4 ticks. This is of course caused by fitness function calculation only at that point. In order to achieve higher performance — i.e. a lower $1/f$ value throughout the evaluation period — we introduce dynamic and random time frames for the evaluation period. This was achieved using 5 different evaluation periods ranging from 8000 to 12000 ticks, in increments of 1000. Three, instead of one, evaluation periods per agent are selected randomly. Then, the fitness value of all three periods is calculated, averaged and assigned to the agent. Those five evaluation period values are chosen based on a compromise between performance and estimated training time.

Repeating the experiment with dynamic evaluation period results in the CR difference values illustrated in Fig. 3. It is

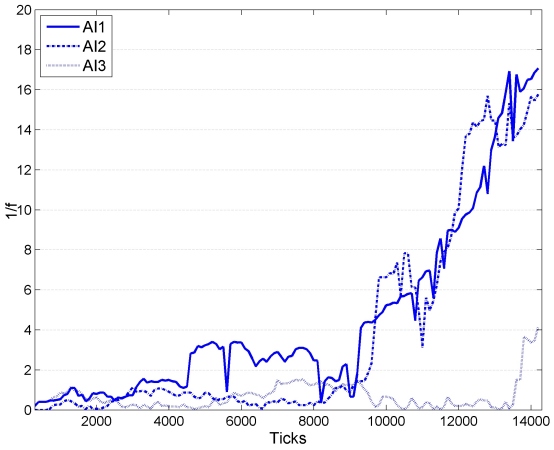

Fig. 3. Dynamic evaluation period; Inverse fitness value ($|\mathrm{CR_{AI}} - \mathrm{CR_o}|$) of three most fit agents during the last evaluation of the training procedure.

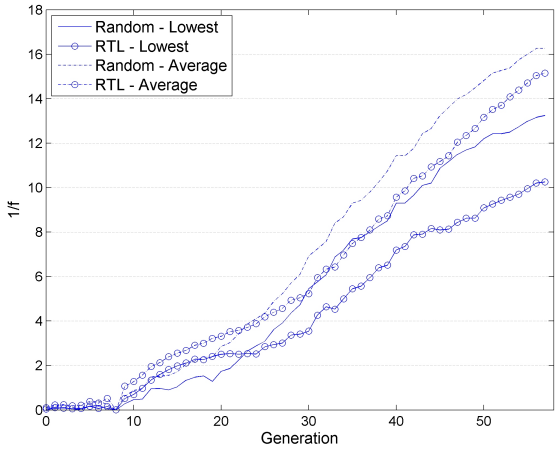

Fig. 4. RTL: Average absolute CR difference $(1/f)$ evolution over generations. Initial ANNs are randomly generated and values presented constitute average values over 10 trials.

now clear that NEAT attempts to minimize the CR difference during the evaluation game period. Compared to the fixed evaluation period experiment, the general shape of the curves displayed indicates a lower CR difference throughout the game, at least until around 11000 ticks. AI 1 and AI 2 agents keep a CR difference lower than 3.5 within the first 9500 game ticks whereas AI 3 keeps the CR difference lower than this threshold until 14000 game ticks are reached. In contrast, fixed evaluation period results revealed a CR difference over 15 even within the first 5000 ticks (see Fig. 2). Moreover, the AI 3 agent appears to perform significantly better than the two other agents illustrated, generating a very smooth curve and very low CR difference variance throughout both the training and the testing phase. Even though the other two agents keep the CR different low during the training phase they fail to generalize when learning is turned off. However, real-time learning via rtNEAT appears to overcome this generalization problem as presented in Section VIII.

VIII. Real-Time Learning

Based on the results presented in Section VII-C, it is clear that the neuro-controlled G2 agent must continuously evolve in order to keep a low absolute CR difference value against an unknown opponent during game play. This could be achieved through real-time learning by the use of rtNEAT.

Real-time learning (RTL) experiments were held on an eight player map scenario: 7 ANN-controlled agents against the standard *Nicowar* agent. The fitness of the population is evaluated every 300 game ticks (approx. 12 seconds of play); this value was determined as a compromise between accuracy of fitness approximation and speed of the RTL mechanism. With a population size of 7, the *ct* value was set to 1 and the game was played for 20000 ticks. A new generation of agents is inserted into the game by replacing existing ANN controlled players using the standard rtNEAT real-time replacement methodology.

As in the NEAT parameter value investigation experiments (see Section VII-A), all 8 players are allied preventing anyone from getting eliminated. This does have the side effect that attack flags (A) are not used in the game, and as such, they do not have an impact on the CR calculation (see eq. (1)). It is our intention to activate and investigate attacking G2 agent behavior in future implementations of RTL.

Fig. 4 shows the RTL curves for the entire population based on this experimental setup. Agents playing following random action-selection are also illustrated for comparison purposes. Following the presentation of off-line learning experiments, the absolute CR difference — rather than the fitness — values are plotted in the graph for a better interpretation. Compared to the baseline random action-selection agents, it appears as if the learning process does diminish the CR differences up to a degree. The difficulty of rtNEAT in constructing efficient ANN controllers within the first 50 generations was expected. The challenging factors which rtNEAT could not effectively overcome are the initial randomly generated agent conditions and the lack of sufficient interaction provided by only 7 agents in the game.

In order to improve RTL performance, the experimental setup was repeated using a previously off-line trained agent: the *WinFast* agent was used as the initial controller of all seven agents playing against *Nicowar*. Again, the experiment was held with RTL turned on and off for comparison purposes. Results of this experiment are shown in Fig. 5.

It is evident that using a well off-line trained agent as a starting point in the search space results in a much higher rtNEAT performance. The performance improvement, albeit rather small (see Fig. 5), is very consistent over 10 training trials. Results indicate that RTL does indeed augment the performance (challenge balance) of ANN controlled agents and that rtNEAT is a reliable method for RTL in an RTS game incorporating G2 features. Nevertheless, the dynamics of the RTL-generated behaviors of G2 agents are not easily

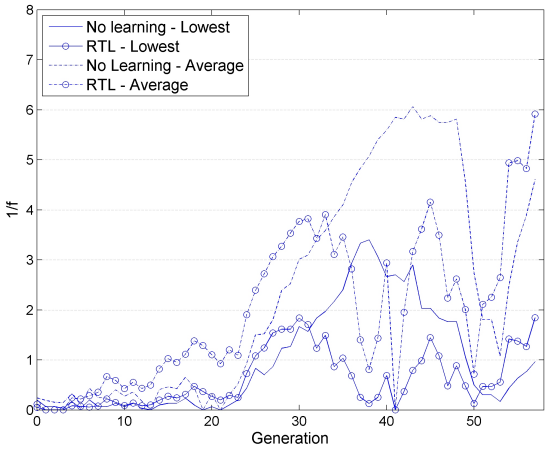

Fig. 5. RTL: Average absolute CR difference $(1/f)$ evolution over generations. Initial agents are controlled by the *Winfast* ANN and values presented constitute average values over 10 trials.

apparent when observing the game.

IX. Conclusions & Discussion

This paper introduces the application of NEAT and rtNEAT to a real-time strategy game, Globulation 2 (G2), for the purpose of dynamic game-challenge balancing. First, we used NEAT to optimize the performance of G2 NPCs and investigate two proposed challenge factors through off-line experiments in G2. Results verified that the factors of *aggressiveness* and *number of warriors* contribute to challenge since obtained neuro-evolved agents were able to outperform all standard AI NPCs available for playing the game.

As a step further, a challenge rating formula was designed incorporating six proposed challenge factors and rtNEAT was used to minimize the difference of challenge rating between the AI agent and its opponent in real-time. Results provide evidence for the effectiveness of the real-time learning and suggest that rtNEAT can be successfully applied for dynamic game balancing in RTS games that share features with G2.

The successful application of rtNEAT in NERO was in a game environment with sufficient interaction and fixed initial conditions for the population. There are 50 soldiers (agents) used as a population in the NERO game and all of them are spawned at the same position. Offspring generated through evolution re-spawn at that same position too. On the other hand, due to G2 game design, there is a maximum of 7 agents to interact with one opponent and offspring are placed in the position of the eliminated members of the population and inherit their game position and current state. This constitutes a significant difficulty in using rtNEAT in a game such as G2.

For future work, we believe that rtNEAT needs to be tested on more complex levels in order to provide more evidence for its generality, and the challenge rating measure proposed needs to be cross-validated against human players. Future rtNEAT implementations extend to off-line parallel processing. Since training is 40 times faster than the speed of the actual game play, several generations of a population could be trained off-line for short periods while the game is running and result to faster adaptation.

While rtNEAT is a great platform for real-time learning, it is probably better suited to games embedding more interaction between the player and the opponents, such as first person shooters, fighting games or 2D action games. For RTS games, a relatively simple temporal difference (TD) learning method like Q-learning could probably perform reasonably well. However, using TD learning with a static ANN structure could hinder advanced behaviors from emerging. Additionally, determining the reward function could be a tedious task in such games.

References

[1] A. J. Champandard, *AI Game Development*. New Riders Publishing, 2004.

[2] K. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[3] K. Stanley, B. Bryant, and R. Miikkulainen, "Real-time evolution in the NERO video game," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, G. Kendall and S. M. Lucas, Eds., Essex University, Colchester, UK, 4–6 April 2005, pp. 182–189.

[4] T. W. Malone, "What makes computer games fun?" *Byte*, vol. 6, pp. 258–277, 1981.

[5] M. Csikszentmihalyi, *Flow: The Psychology of Optimal Experience*. New York: Harper & Row, 1990.

[6] ——, *Beyond Boredom and Anxiety: Experiencing Flow in Work and Play*. San Francisco: Jossey-Bass, 2000.

[7] N. Lazzaro, "Why we play games: Four keys to more emotion without story," XEO Design Inc.," Technical Report, 2004.

[8] G. N. Yannakakis and J. Hallam, "Towards Optimizing Entertainment in Computer Games," *Applied Artificial Intelligence*, vol. 21, pp. 933–971, 2007.

[9] G. Andrade, G. Ramalho, H. Santana, and V. Corruble, "Extending reinforcement learning to provide dynamic game balancing," in *Proceedings of the Workshop on Reasoning, Representation, and Learning in Computer Games, 19th International Joint Conference on Artificial Intelligence (IJCAI)*, August 2005, pp. 7–12.

[10] M. A. Verma and P. W. McOwan, "An adaptive methodology for synthesising Mobile Phone Games using Genetic Algorithms," in *Congress on Evolutionary Computation (CEC-05)*, Edinburgh, UK, September 2005, pp. 528–535.

[11] R. Hunicke and V. Chapman, "AI for Dynamic Difficulty Adjustment in Games," in *Proceedings of the Challenges in Game AI Workshop, 19th Nineteenth National Conference on Artificial Intelligence (AAAI'04)*, 2004.

[12] P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma, "Difficulty Scaling of Game AI," in *Proceedings of the 5th International Conference on Intelligent Games and Simulation (GAME-ON 2004)*, 2004, pp. 33–37.

[13] J. Ludwig and A. Farley, "A learning infrastructure for improving agent performance and game balance," in *Proceedings of the AIIDE'07 Workshop on Optimizing Player Satisfaction, Technical Report WS-07-01*, G. N. Yannakakis and J. Hallam, Eds. AAAI Press, 2007, pp. 7–12.

[14] R. Koster, *A Theory of Fun for Game Design*. Paraglyph Press, 2005.

[15] X. Yao, "Evolving artificial neural networks," in *Proceedings of the IEEE*, vol. 87, no. 9, 1999, pp. 1423–1447.

[16] K. Stanley and R. Miikkulainen, "Efficient Reinforcement Learning through Evolving Neural Network Topologies," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2002, pp. 569–577.

[17] J. K. Olesen, "Optimizing game challenge using real-time adaptive artificial intelligence," Master's thesis, University of Southern Denmark, 2008.

[18] "Globulation 2," Free software RTS game with a new take on micro-management, 2008, available at http://www.globulation2.org/.