Evolutionary Dynamic Scripting: Adaptation of Expert Rule Bases for Serious Games

Reinier Kop[1], Armon Toubman[2], Mark Hoogendoorn[1], Jan Joris Roessingh[2]

[1] VU University Amsterdam, Department of Computer Science
De Boelelaan 1105, 1081HV Amsterdam, The Netherlands
{r.kop, m.hoogendoorn}@vu.nl

[2] National Aerospace Laboratory NLR
Anthony Fokkerweg 2, 1059CM Amsterdam, The Netherlands
{Armon.Toubman, Jan.Joris.Roessingh}@nlr.nl

Abstract. Automatically generating behavior for Non-Player Characters (NPCs) in serious games can be problematic as the specification of their behavior heavily relies on the availability of domain expertise. This expertise can be difficult and costly to extract, and the specified behavior usually does not allow for generalization to new scenarios or users. Alternatively, behavior can be generated using a pure machine learning approach. However, such NPCs may quickly develop static, non-adaptive behavior by exploiting the environment without proper constraints. In this paper, an approach called Evolutionary Dynamic Scripting (EDS) is presented to effectively cope with the disadvantages of the two extremes sketched above. This technique combines the generative characteristics of an evolutionary approach with an adaptive reinforcement learning method called Dynamic Scripting. Dynamic Scripting essentially learns how to prioritize rules from a fixed rule-base specified by domain experts. EDS was tested in an air combat simulation in which agents co-evolve their tactics using EDS. EDS was able to generate improved behavioral rules over the original Dynamic Scripting approach, given the same initial rule-bases. Both generalization to new situations and specialization into roles for the agents were observed.

Keywords: serious gaming, dynamic scripting, evolutionary algorithms

1 Introduction

In the last decade, serious gaming approaches to (military) training applications have gained widespread popularity. Serious games have a number of benefits over 'classical' training, such as cost reduction, the possibility of training with more individualized scenarios, and the ability to produce events that cannot easily be staged in real life.

In many serious games, the behavior of the Non-Player Characters (NPCs) is of crucial importance. Whether filling the role of teammates, tutors, or adversaries,

NPCs should exhibit dynamic, adaptive behavior. However, generating such behavior often requires complex models based on expert knowledge (see e.g. [1]). Creating expert models for a sufficiently large number of training scenarios can be cumbersome. As a result, these models often do not exhibit a sufficiently rich palette of behaviors, and hence result in a highly predictive learning experience. An alternative is to deploy machine learning to generate appropriate behavior models from scratch (see e.g. [2]). Still, guaranteeing dynamic behavior remains problematic.

Several approaches that try to combine the best of both worlds have been proposed, among which the Dynamic Scripting (DS) technique [3]. DS is a reinforcement learning (RL) technique that starts with a set of generally applicable behavior rules provided by domain experts, and lets agents learn how to prioritize these rules for a specific scenario in a series of RL trials. Favoring adaptation speed over optimal performance, this method is not like traditional RL methods, which attempt to create a policy (a mapping of optimal actions to take given a situation). DS heavily relies on the assumption that the initial rule set provides 'good-enough' rules for all possible scenarios, which might not always be the case as the need for additional rules might occur in novel scenarios.

In this paper, a technique is presented which is called Evolutionary Dynamic Scripting (EDS). In EDS, the DS learning process is embedded in an evolutionary method which enables the discovery of new rules. In essence, DS serves as the fitness evaluation function for the evolutionary method. This combination allows for more flexibility in the generation of specific behavior for novel scenarios, while using more general rules designed by a domain expert as a starting point. This way, appropriate domain knowledge is utilized and new behavior is generated where it is shown to be effective, while reducing the workload of the domain expert. The approach is evaluated in an air combat simulation [4], as several DS case studies have already been performed in this domain and can be used as a benchmark.

This paper is organized as follows. Section 2 sketches the relevant background of this work. The EDS approach is described in Section 3 and the experimental setup is described in Section 4. Section 5 presents the results, and finally, Section 6 concludes the paper.

2 Background

A variety of techniques have been proposed that enable the generation of NPC behavior based on domain expertise. Many of these techniques come in the form of cognitive models that stem from human decision-making processes. These techniques mainly focus on the realism of the generated behavior. Swartout *et al.* [1] for example introduce an architecture for virtual reality-based training in which the virtual agents use task models to reason about causality and the distribution of tasks between human and virtual agents. In [5], Merk presents a number of cognitive models, addressing various aspects of fighter pilot behavior. These models were validated in evaluations in which fighter pilots received training in simulators using enemies driven by the cognitive models. Methods that are still cognitive-based but do add some anticipation

elements include theory of mind based approaches (see e.g. [6] and [7]). However, these approaches all heavily rely on domain expertise.

On the other side of the spectrum, various approaches are based on pure machine learning techniques to establish adaptive NPC behavior. A complete overview of this domain is beyond the scope of this paper, but examples for the domain of fighter pilot behavior can be found in [2]. Bellotti *et al.* [8] introduce an agent based on machine learning that is able to adapt the flow of a serious game during play.

The current research is not the first attempt to combine expert knowledge with machine learning techniques. As said, the DS technique is based on the prioritization of rules depending on their appropriateness for the situation at hand. An alternative approach that has been proposed is to tailor a cognitive model to the situation at hand by means of adapting the parameters of the model, also allowing for a form of adaptation [9]. Although these techniques provide ways to adapt existing behavior to new scenarios, they do so in a relatively limited way as they are based on existing rules. Generating new rules is the focus of the work presented here, thereby still taking advantage of information obtained from domain experts. An additional advantage of using and adapting behavior rules is that all NPC behavior is defined in a human-readable way. After the machine learning process, a domain expert or training instructor can always review the learned behavior and make manual changes, if needed.

3 Method

EDS attempts to improve NPC behavior by repeatedly evaluating the performance of an agent's rules using DS, and evolving the evaluated rules using an evolutionary method based on genetic programming (GP). Implementing such a system has two major advantages. First, as with regular DS, the use of behavior rules provides transparency throughout the learning process, as the behavior model is always human-readable. Second, it decreases the need for a domain expert, since the evolutionary method is able to optimize existing rules, or even discover completely new rules.

EDS can be classified as a specialized Learning Classifier System (LCS). LCSs attempt "to evolve a system that will respond to the current state of its environment" [13]. An LCS is usually based on optimizing a policy, using a genetic algorithm (GA) and a reinforcement component. Instead of a GA, EDS uses GP and instead of traditional RL for the reinforcement component, EDS uses DS. EDS is thus able to use rule structures, rather than binary sequences. Moreover, it doesn't necessarily have to create rules from scratch (such as the LCS used in [2]), but is able to optimize existing rules, which it can evaluate rapidly.

The main EDS loop is shown in Algorithm 1. The algorithm is initialized with an initial rule base that is either predefined or generated from scratch. Then, the evolutionary loop starts (lines 3). First, the fitness (or expected effectiveness) of each individual rule is evaluated in the *DS component* (starting in line 4). In the DS component, the rule base is optimized by DS, which results in a fitness value for each rule. The fitness may be revaluated multiple times at each generation (lines 4-6), allowing (weighted) averaging of a rule's fitness values across multiple DS *learning episodes*,

thereby increasing the robustness of EDS. Once the fitness is known and stored (line 6), the *GP component* alters various rules in the rule base based on their fitness (shown in line 7). This constitutes one *generation*, after which the (new) rule base can again be evaluated using the DS component. This loop terminates when some termination condition is met (such as a maximum number of generations).

Algorithm 1. EDS

```
1:  rule_base ← initial_rule_base
2:  results ← array[max_episodes]
3:  for generation ← 1 to max_generation do // EDS loop
4:    for episode ← 1 to max_episode do // DS loop
5:      results[episode] ← perform_DS(rule_base)
6:    fitness ← evaluate_fitness(result)
7:    rule_base ← evolve(rule_base, fitness) // GP component
```

3.1 Dynamic Scripting Component

DS is a form RL which allows an autonomous agent to dynamically adjust its behavior based on feedback from the environment [3]. DS was originally designed for computer role-playing games, but it has since been adapted for usage in other genres such as real-time strategy games [10, 11], first-person shooters [12], and air combat simulators [4]. DS is not like other RL methods because it requires predefined behavioral rules which are not modified during the learning process. While agents using DS are unable to find new behavior, it also guarantees that agents cannot learn behavior that is worse than the predefined rules. Another difference with DS and traditional RL is the fact that DS does not attempt to make a mapping between observed states and desired actions. This makes it easier for DS to (dynamically) adapt to new situations.

Each agent using DS maintains a set of predefined rules in its rule base, together with a weight value for each rule. For every encounter, a subset of rules is stochastically selected from the rule base, directly proportional to the weight value for each rule.

The selected rules form a script that is used to control the agent during an encounter. After each encounter, the DS algorithm adjusts each rule's weight value based on the outcome of the encounter. If the rules in the script performed well during the encounter, their weights are increased; conversely, if the rules in the script performed badly during the encounter, their weights are decreased. The weight value of each rule therefore comes to represent the expected effectiveness of the rule against the current opponent(s). This redistribution of weights leads to reselection of favorable rules.

After a sufficient number of encounters, we treat the weight of each rule as its fitness value for use in the GP component. When evolving a large set of rules (as we are trying to accomplish with EDS), DS has the advantage of being able to evaluate subsets of the rules in various specific circumstances. Which subsets are applicable under what circumstances is detected automatically by the DS algorithm. At the same time, because only one large set of rules is being evolved, it is possible to maintain a level of general applicability.

3.2 Genetic Programming Component

In the GP component (line 7 in Algorithm 1), the rule adaptation takes place. Once the rules' fitness values are known, we adapt the rules using evolutionary principles, i.e. offspring is created and survivors are selected. The steps of the GP component are shown in Algorithm 2.

Algorithm 2 | evolve(rule_base, fitness)

```
1:  child_num ← 0
2:  while child_num < max_children do
3:      parents ← select_parents(rule_base, fitness)
4:      if random() < prob_crossover then
5:          children ← crossover(parents)
6:      else
7:          children ← mutate(parents)
8:      for child in children do
9:          rule_base ← survivors(rule_base + child)
10:     child_num ← child_num + 2
11: return rule_base
```

Parent selection (line 3) is done through fitness proportionate selection. Two parents are selected per cycle. After selection, crossover is applied with a probability *prob_crossover* (lines 4-5), or else they are mutated (lines 6-7). Applying *either* mutation *or* crossover (as opposed to both) is a common procedure for GP [13].

In lines 8-9, the newly generated children are inserted in the rule base. After adding a child, *survivors* are immediately *selected*. The rule with the lowest fitness value is deterministically removed.

Genetic Operators

To facilitate the use of genetic operators, the rules in the rule base are represented as tree structures. The crossover operator used is subtree crossover, which creates new children by randomly exchanging subtrees of two parents. Each of the subtrees is selected with a probability $p = 1/n$, where n is the number of expressions in the rule.

The mutation operator is one of three methods, chosen randomly (with equal probabilities): point, subtraction, and addition mutation. In point mutation, each expression in a rule has a probability $p = 1/n$ of changing to another random expression, where n is the number of expressions in the rule. Subtraction mutation randomly removes an entire subtree from the rule. This subtree is selected identical to the aforementioned crossover method. With equal probabilities, addition mutation either takes a subtree subtracted by a previously applied subtraction mutation, or randomly generates and adds a subtree to the rule. This random generation is done using a simple grammar in which all valid tree structures are expressed. This grammar prevents invalid rules and erroneous combinations from being created, such as 'fire a missile at an ally'. The choices made in the grammar are always made with equal probabilities.

After performing these steps, the contents of the rule base will have changed, meaning each rule's fitness can be evaluated again using the DS component. This evaluation also includes existing rules that survived the previous generations.

4 Experimental Setup

To investigate the proposed method from various angles, we split the experiments in three stages: rule base generation, validation, and generalization. These stages and the scenario in which they are applied are explained below.

Scenario.

A simulated scenario was used (identical to the scenario in [4]) in which two F-16 aircraft (the blues), a 'flight lead' and a 'wingman', engage an enemy F-16 aircraft (the red). The latter is performing a so called Combat Air Patrol (CAP), i.e. repeatedly flying a circular pattern in the airspace that it needs to defend (see Figure 1). Each simulated encounter, the team that first eliminates an aircraft from the other team wins. The blue aircraft are controlled by agents whose rule bases are generated using the EDS approach. The red aircraft is controlled by an agent that uses one of six tactics:

- t_{default} represents the default opponent's tactic. Red flies a counter-clockwise CAP, and fires at enemies upon detection.
- t_{evading} is as t_{default}, but includes evasive maneuvers.
- $t_{\text{close\_range}}$ is as t_{default}, but fires missiles from a shorter range.
- $t_{\text{default\_alt}}$, $t_{\text{evading\_alt}}$, $t_{\text{close\_range\_alt}}$ are as t_{default}, t_{evading}, $t_{\text{close\_range}}$, respectively, but flies the CAP in a clockwise fashion.

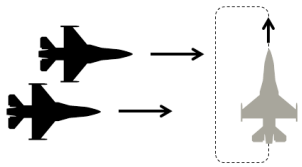

Fig. 1. The two blues (left) try to intercept red (right), who is flying a Combat Air Patrol.

Rule Base Generation.

In this experimental stage, EDS is applied to the rule bases of both blues, in an attempt to generate new, improved rule bases. This experiment is meant to show that EDS can find rules that are competitive to rules provided by experts, provided that a reasonably rich initial rule base is present. For the used virtual environment, an Expert Rule Base (ERB) for each blue agent is available and known to be working well with respect to the described scenario. However, we cannot be sure whether these ERBs can be further improved by EDS (it may be the ERBs are already the best possible set of rules for this scenario, given that an expert has specified it). To be certain the initial rule bases are non-optimal, we weaken the ERBs with respect to their originals. This provides us with a Degenerated Rule Base (DRB) for each blue agent.

Both agents' DRBs are evolved concurrently against each of the opponent's tactics. Each application of EDS is performed thirty generations, with ten DS learning episodes per generation, and fifty encounters per learning episode. Ten EDS applications are run per tactic of red. These parameters are based on prior tests.

Each generation, each blue agent generates new rules based on old ones using the mutation operator only. This was empirically found to outperform the use of both crossover and mutation. This predominantly results in exploitation of the rule base

(associated with mutation), as opposed to exploration (associated with crossover). When mutation is applied, one of the three mutation operators is randomly chosen.

The performance of the blues in this phase is the ratio of blue wins with respect to the total number of performed encounters.

Validation.

The output of the rule generation phase is six sets of rule bases per blue agent (i.e. one per enemy tactic). Ideally, we would have a single rule base which is able to adapt to multiple tactics, the big advantage of regular DS. To achieve this, rules of the rule bases obtained from the previous section are combined so that we end up with a Combined Rule Base (CRB) for each blue agent, equal in size to the ERB and the DRB. The CRB is made by taking the inclusive disjunction of the each of the tactic's best performing rule base from the previous section, ordered by fitness. This set is cut off at the correct rule base size mark (31 for the flight lead, 32 for the wingman).

To validate whether EDS has had any significant impact on the blue's rule bases, the performance of the DRB, ERB, and CRB against each of the six opponent's tactics are all measured using regular DS (since we just want to evaluate the performance of rule bases resulting from EDS). The significance of this comparison is as follows: if the CRB outperforms both other rule bases, we can conclude that the rules evolved by EDS in some way improved the behavior of the ERB, which was one of the goals of this research. We hypothesize that the CRB will outperform the DRB (hypothesis 1), and perform at least as well as the ERB (hypothesis 2) when applying regular DS on either. Hypothesis 2 is a bit more conservative as ERB might already be (near) optimal. The blue agents are trained during 100 learning episodes against each tactic. Each learning episode simulates 250 encounters. During this stage, performance is measured as the running average of the current win ratio (window size 20), averaged over the 100 learning episodes. The large number of learning episodes is chosen to increase the likelihood of the hypothesized effect, given the stochastic nature of both the simulation environment and the DS/EDS techniques.

Generalization.

To study the generated rules' ability to generalize, we combine CRBs using five out of the six evolved rule bases. This is done for each combination of five rule bases against its respective previously unseen tactic. For example, we can measure $CRB_{\sim default}$'s performance (with the ~ representing 'not'), consisting of all evolved rule bases except for the one evolved against $t_{default}$, against tactic $t_{default}$. As such, six partially combined rule bases $CRB_{\sim n}$ are generated per blue agent, where n indicates an adversary's tactic. Generating a partially combined rule base is done in a fashion identical to how the CRB was generated. Similar to the previous experimental phase, each of these partially combined rule bases is then used for running DS against each tactic. Again, we apply 100 learning episodes, and 250 encounters per learning episode, where the performance is the running average of window size 20. We hypothesize that the regular CRB will outperform each $CRB_{\sim n}$ using regular DS for both cases (hypothesis 3), because it has complete knowledge of the current opponent's tactic.

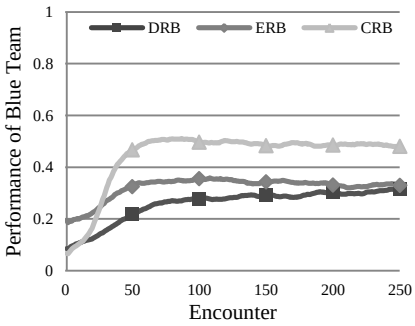

Fig. 2. Results of DS with various initial rule bases, averaged over all tactics, (100 learning episodes per tactic).

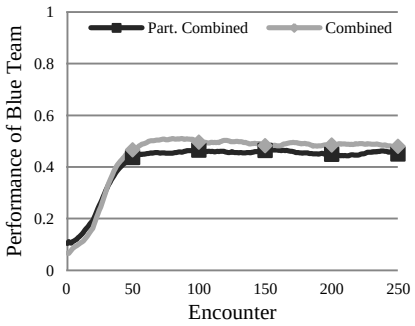

Fig. 3. Results of DS with fully and partially combined rule bases, averaged over all tactics (100 learning episodes per tactic).

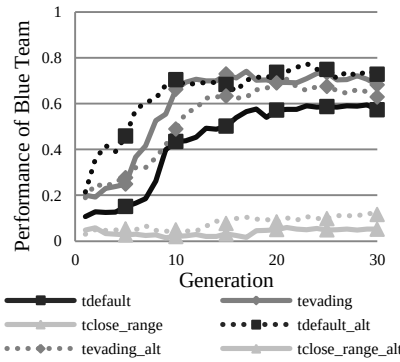

Fig. 4. Average of 10 EDS runs for each tactic played against.

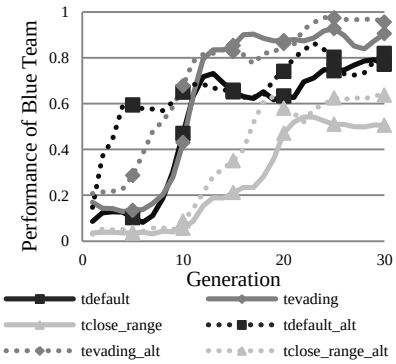

Fig. 5. Best of 10 EDS runs for each tactic played against.

5 Results

Rule Base Generation.

Figure 2 shows the average over ten EDS runs against each tactic. The runs against t_{close_range} and $t_{close\_range\_alt}$ evidently perform much worse relative to the runs against the other four tactics. The average final performance per tactic is highest against $t_{default\_alt}$ (0.728), followed by $t_{evading}$ (0.682), $t_{evading\_alt}$ (0.629), $t_{default}$ (0.573), $t_{close\_range\_alt}$ (0.116), and t_{close_range} (0.052).

Figure 3 shows the best EDS run out of 10. The lines in the figure are smoothed using a running average with a window size of 3. t_{close_range} and $t_{close\_range\_alt}$ are in fact able to reach a relatively high performance in a few cases. The best performance per tactic in the final generation is greatest against $t_{evading\_alt}$ (0.956), followed by $t_{evading}$ (0.905), $t_{default\_alt}$ (0.819), $t_{default}$ (0.771), $t_{close\_range\_alt}$ (0.637), and t_{close_range} (0.506).

The results show that EDS is able to generate rules of increasing quality against four out of six tactics. This is in line with the research reported in [4], where the blues

against t_{close_range} and $t_{close\_range\_alt}$ were found to perform worse when applying DS. This suggests that the close range tactics are simply more difficult to win against.

When exploring the behavior exhibited by the newly generated tactics, a limited form of role specialization was observed. For example, one agent would behave as an engaging agent, with rules describing how to engage the opponent. Another agent behaved as an evading agent, with mostly rules of evasion. After the experiment, the engaging agent had many near-identical rules, e.g. a rule for firing missiles when no other missile is flying towards the target, and another for firing missiles only if the agent's wingman has no missiles left. A similar situation held for the evading agent. This indicates a certain degree of convergence has taken place for these rule bases.

Validation.

The performances of DRB, ERB, and CRB have been averaged over the different opponent tactics. An independent two-tailed t-test assuming unequal variances (α = 0.05) shows that the average performance after the final encounter of the CRB is significantly higher than both DRB ($p << 0.05$) and ERB ($p << 0.05$) performances, as suggested in Figure 4. This confirms hypothesis 1 and 2, suggesting the CRB (and thus EDS) make a significant contribution to the scenario. Additionally, the ERB and DRB performances do not differ significantly from each other ($p = 0.513$).

Generalization.

To investigate generalizability, again the performances for all tactics are averaged, providing us with results for which the relevant adversary tactic was used during the learning process to form the rule set (fully combined), and results for which it was not (partially combined). A two-tailed t-test assuming unequal variances ($\alpha = 0.05$) shows there is no difference in performance ($p = 0.086$), as Figure 5 suggests. Thus, hypothesis 3 is rejected, suggesting a certain measure of redundancy was introduced when evolving rule bases against different tactics. Finally, the graphs in Figure 4 and Figure 5 stabilize at the 0.5-mark. This is likely due to chance; the red and blue team setups are different from one another. Slight changes would quickly shift this equilibrium.

6 Conclusion

Though NPCs are rapidly becoming more intelligent, there is still a long way to go before they can match human intelligence. EDS attempts to partially bridge this gap by focusing on the adaptivity by evolving rule bases, while at the same time reducing the required domain knowledge. We investigated whether EDS could be an improvement relative to DS in terms of behavior, domain expertise, and generalization.

We showed that EDS is able to generate improved behavior. Simpler and sometimes more specialized rules were found. This suggests that EDS may be able to reduce the workload of domain experts, since they no longer have to focus on creating different roles for NPCs; they can simply design a generic set of rules, after which

EDS automatically assigns certain roles to certain NPCs. Whether the observed generalizability of the rules also holds in other scenarios should be investigated.

Regarding future work, there are a number of improvements and interesting possibilities left to explore. First, the realism of the behavior in the system needs to be investigated. Second, generating rules from scratch as opposed to using a predefined rule base may create more diverse behavior. Finally, since a rule's fitness is evaluated in the context of script and not in isolation, it is worthwhile to investigate how to incorporate rule's interactions with each other when assigning fitness values.

References

[1] W. Swartout, J. Gratch, R. Hill, E. Hovy, S. Marsella, J. Rickel en D. Traum, „Towards Virtual Humans," *AI Magazine vol. 27,* pp. 96-108, 2006.

[2] R. Smith, A. El-Fallah, B. Ravichandran, R. Mehra en B. Dike, „The Fighter Aircraft LCS: A Real-World, Machine Innovation," *Applications of Learning Classifier Systems,* pp. 113-142, 2004.

[3] P. Spronck, M. Ponsen, I. Sprinkhuizen-Kuyper en E. Postma, „Adaptive game AI with dynamic scripting," *Machine Learning 63.3,* pp. 217-248, 2006.

[4] A. Toubman, J. J. Roessingh, P. Spronck, A. Plaat en J. Van den Herik, „Dynamic Scripting with Team Coordination in Air Combat Simulation," in *Proceedings of the 27th International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems*, Kaohsiung, Taiwan, 2014.

[5] R.-J. Merk, Cognitive Modelling for Opponent Agents in Fighter Pilot Simulators, Agent Systems Research Group, VU University, Amsterdam: Ph.D. Thesis, 2013.

[6] M. Harbers, K. Van den Bosch en J. Meyer, „Modeling Agent with a Theory of Mind," in *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, 2009.

[7] M. Hoogendoorn en J. Soumokil, „Evaluation of Virtual Agents Attributed with Theory of Mind in a Real Time Action Game," in *In: van der Hoek, Kaminka, Lesperance, Luck, and Send (eds.), Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems*, 2010.

[8] F. Bellotti, R. Berta, A. de Gloria en L. and Primavera, „Adaptive Experience Engine for Serious Games," *IEEE Transactions on Computation Intelligence and AI in Games, vol. 1(4),* pp. 264-280, 2009.

[9] R. Koopmanschap, M. Hoogendoorn en J. Roessingh, „Tailoring a Cognitive Model for Situation Awareness using Machine Learning," *Applied Intelligence,* vol. 42, nr. 1, pp. 36-48, 2015.

[10] A. Dahlbom en L. Niklasson, „Goal-Directed Hierarchical Dynamic Scripting for RTS Games," in *AIIDE*, 2006.

[11] M. Ponsen, Improving adaptive game AI with evolutionary learning, TU Delft: PhD Thesis, 2004.

[12] D. Policarpo, P. Urbano en T. Loureiro, „Dynamic scripting applied to a First-Person Shooter," in *5th Iberian Conference on Information Systems and Technologies (CISTI)*, 2010.

[13] A. E. Eiben en J. E. Smith, Introduction to evolutionary computing, Berlin: Springer, 2010.