Rapid Adaptation of Air Combat Behaviour

Armon Toubman[1], **Jan Joris Roessingh**[1], **Pieter Spronck**[2], **Aske Plaat**[3], and **Jaap van den Herik**[4]

Abstract. Adaptive behaviour for computer generated forces enriches training simulations with appropriate challenge levels. For adequate insight into the range of possible behaviour, the adaptation has to take place in a rapid fashion. Ideally, each new behaviour model should remain readable by (and thereby under the control of) human experts. Although various attempts have been made at creating adaptive behaviour, current solutions require large numbers of simulations. Moreover, usability by end users has been of subordinate interest, as is compliance with doctrine and ethics. In this work, we present a machine learning method that enables fast behaviour adaptation, while keeping the behaviour models in a human-readable format. We demonstrate the effectiveness of the proposed method in beyond-visual-range air combat simulations.

1 INTRODUCTION

The use of training simulations for defence applications is growing [1]. Commercial off-the-shelf simulation packages, such as STAGE [2] and Virtual Battlespace (VBS) [3], allow experts to quickly develop and operate scenarios for simulations. To make the scenarios more realistic, they are often inhabited by computer generated forces (CGFs).

Traditionally, CGF behaviour is scripted using if-then rules which map observations to actions. However, writing good scripts requires domain expertise, which is a costly resource. Poorly written scripts have low training value, as no skills learned by the trainee are transferable to the real world. Furthermore, trainees can learn to purposefully exploit bad CGF behaviour. This is usually counterproductive and should be discouraged [4, 5]. Dedicated CGF behaviour authoring tools, such as Smart Bandits [6], have been developed to mitigate this issue, often by introducing enhanced user interfaces and ready-to-use behaviour modules. However, it is still up to the experts to design the CGF behaviour and adapt this behaviour to reach specific training goals.

Nowadays, advances in the field of machine learning offer the prospect of automatically generating behaviour models and adapting these models online (i.e., during operation in training simulations). Automatic generation of behaviour models has the potential to greatly decrease the workload of CGF developers, while online adaptation can increase the training value of CGFs by continuously challenging the trainees. Over the years, various machine learning approaches have been tried, yet adaptive capabilities in CGF behaviour authoring tools are still rare [5, 7]. Adoption of these approaches is tamed by the large amount of time needed for quality control: proposed machine learning methods for CGF behaviour generation require substantial processing times, and produce behaviour models that are hard for human end users to understand. The latter is a critical feature, as end users need to be able to verify that generated behaviour complies with doctrine, and by extension, ethics. This consideration is also important in related fields, e.g., the development of behaviour for autonomous unmanned vehicles, or decision support systems for human pilots.

In this work, we present a machine learning method that is specifically focused on rapid generation of understandable behaviour models. The method entails the adaptation of behaviour represented as finite-state machines (FSMs), through a reinforcement learning technique called dynamic scripting.

FSMs have been successfully used to represent CGF behaviour in the Smart Bandits behaviour authoring tool, which is currently in use by the Royal Netherlands Air Force (RNLAF) to control CGFs in beyond-visual-range air combat training simulations. By cutting up the FSMs into their constituent states and transitions, the dynamic scripting algorithm is able to efficiently recombine the FSMs and provide adaptive behaviour. Furthermore, in contrast to many other machine learning algorithms, dynamic scripting does not alter defined pieces of behaviour during the learning process, which is a great step towards keeping generated behaviour in line with military doctrine, and keeping behaviour models understandable by experts.

To the best of our knowledge, this is the first work developing adaptive capabilities for two cooperative CGFs in 2v2 beyond-visual-range air combat. In this work, we actively take into account (1) computational speed, (2) usability by end users, and (3) built-in ethical and doctrinal consideration.

The rest of this paper is structured as follows: Section 2 gives an overview of related work. Section 3 describes the integration of FSMs into dynamic scripting. Section 4 shows the experimental setup used to test the adaptive CGFs. The results of the experiments are presented in Section 5 and discussed in Section 6. Finally, section 7 concludes the paper.

2 RELATED WORK

Air combat is the fight between armed aircraft. It can be represented as a ‘game’ with a large, continuous state space, a variety of available actions, and limited resources (see [8] for a complete treatise). When generating air combat behaviour, creative solutions are required while being bound by tactical doctrine and rules of engagement (and training goals, for training simulations).

[1] Department of Training, Simulation & Operator Performance, Netherlands Aerospace Centre, Netherlands, email: {Armon.Toubman,Jan.Joris.Roessingh}@nlr.nl
[2] Tilburg center for Cognition and Communication, Tilburg University, Netherlands, email: p.spronck@gmail.com
[3] Leiden Institute of Advanced Computer Science, Leiden University, Netherlands, email: aske.plaat@gmail.com
[4] Leiden Centre of Data Science, Leiden University, Netherlands, email: jaapvandenherik@gmail.com

Air combat is usually divided into within-visual-range (WVR) combat (also known as air combat manoeuvring or dogfighting) and beyond-visual-range (BVR) combat, in which combating aircraft engage each other with long-range sensors and weapons. WVR and BVR combat require different approaches: WVR is often modelled as a pursuit-evasion problem, consisting of complex manoeuvring and rapid decision-making, whereas BVR requires planning and higher-level strategical thinking.

2.1 Machine learning for air combat behaviour

A wide range of machine learning techniques has been tried to efficiently generate effective WVR and BVR air combat behaviour. A non-exhaustive overview of these approaches is given below. The research in this area is quite fragmented, not only between WVR and BVR combat, but also between simulation environments and experimental methods. While this means that no absolute comparisons can be made among reported results, the reported parameters may serve as an indication of the computational complexity of the methods.

Neural networks have been applied in various ways with varying success. Early work with neural networks includes the use of a three-layer back-propagation network by Rodin and Amin [9] for predicting and countering WVR tactical manoeuvres. Rodin and Amin report "successfully training" their network after 60,000 iterations. More recently, Teng et al. [10] applied self-organizing neural networks with a Q-learning component for online learning of WVR behaviour. The resulting behaviour models were evaluated in small-scale human-in-the-loop experiments. The learning network was able to reach a 93% mean win ratio after 120 episodes against a statically controlled CGF. Furthermore, the network peaked at a 40% win ratio against pilots in training, and below 10% against experienced pilots. Teng et al. report using available air combat doctrine for building the state- and action-space for the Q-learning component [11], by encoding expert knowledge as if-then rules.

Evolutionary algorithms have also been used in various forms. Mulgund et al. [12] applied a genetic algorithm to optimize tactical parameters for many-versus-many BVR engagements. Starting from a scenario with equal losses on both sides, their algorithm was able to develop tactics by which all enemy CGFs were defeated, without any casualties on the friendly side. However, only few parameters are reported. In a follow-up study, Zhang et al. [13] used 40 generations, with a population size of 80. Smith et al. applied a learning classifier system to develop novel one-versus-one WVR tactics for an experimental fighter jet [14, 15]. A population of 200 rules is reported, tested throughout 300 generations. Furthermore, air combat tactics have been described through grammars, which have been used as templates for genetic programming algorithms (see, e.g., [16] and [17], both BVR). Expressing tactics through grammars limits the search space, ensuring that only valid behaviour is generated. However, large numbers of simulations are seemingly needed to reach convergence using this method, with for example [16] reporting convergence near 50% fitness after 100,000 simulations.

While a large number of simulations may be acceptable for exploratory studies such as [15], or offline learning before human-in-the-loop trials, it poses a problem in the case of learning online during training simulations. A CGF, trying to adapt its behaviour to that of a human participant, only has limited time to do so between engagements. Furthermore, trainees can only participate in a limited number of simulations, which constrains the time available to adapt even further.

2.2 Transparency of behaviour

Apart from the time to adapt, the transparency of generated behaviour models is of great importance. Behaviour models generated for military applications should be usable (editable, readable, testable, etc.) by different end users, e.g., scenario developers and training instructors [7]. Techniques such as neural networks and evolutionary algorithms often produce behaviour models that are hard to decode, understand, and manually edit.

In earlier work, we have made attempts at generating BVR air combat behaviour using dynamic scripting [18, 19]. Dynamic scripting is a reinforcement learning method that takes a rule base, and recombines the rules from this rule base into scripts [20]. This method does not 'invent' new behaviour, and instead relies on the rule base being filled with rules based on expert knowledge. As a result, the generated behaviour can only be as good or bad as the knowledge contained in the rule base. Applying a pure dynamic scripting solution in the air combat domain has yielded encouraging results, however the technique remains to be validated in a production environment.

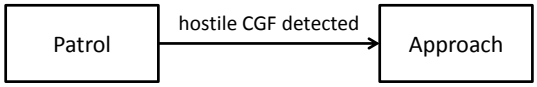
Rather than having experts write if-then rules, a more intuitive method of defining behaviour is the use of finite-state machines [21]. This is also the method used in Smart Bandits [6, 22, 23], the CGF behaviour authoring tool developed by the Netherlands Aerospace Centre, and currently in use by the Royal Netherlands Air Force. Each CGF controlled by Smart Bandits is in a certain state, and each state has associated actions. However, Smart Bandits provides no adaptive capabilities. As Smart Bandits provides both (1) a drag-and-drop interface for authoring CGF behaviour, usable by various end users, and (2) an established repository of well-tested CGF behaviour that is actively being used in training simulations, it is an ideal testing ground for introduction of adaptive behaviour.

3 ADAPTIVE FINITE-STATE MACHINES

In Smart Bandits, CGFs are controlled with FSMs. When FSMs are to control CGF behaviour, the states of the FSM are linked to pieces of related behaviour [21]. For example, a *Patrol* state may correspond to a CGF repeatedly moving between two points in the simulated world (see Figure 1a). A transition to another state then occurs when a certain change in the world state is perceived by the CGF. Continuing the example, if the CGF is in the *Patrol* state and detects a hostile CGF, it might transition to the *Approach* state in which the CGF will move towards the detected CGF. The example above can be expressed as a set of rules, as shown in Figure 1b.

The resulting rules can now be stored in a rule base, which serves as the input for the dynamic scripting technique. As mentioned in Section 2, dynamic scripting [20] is a rule-based reinforcement learning technique. When the dynamic scripting algorithm is initialized with a rule base, it assigns a weight value to each rule in the rule base. Before each episode (in our case, a simulated air combat encounter), a predefined number of rules are drawn from the rule base through roulette wheel selection, in which each rule is represented by its weight. Together, the rules that are drawn from the rule base form the script that governs the behaviour

(a)




```
if state == Patrol:
   if near(point_a): move_to(point_b)
   if near(point_b): move_to(point_a)
```

(b)

```
if state == Approach:
  move_to(detected_hostile_CGF)

if state == Patrol and hostile_CGF_detected():
     set_state(Approach)
```

Figure 1. Representing a behavior controller as a finite-state machine, (a) graphically and (b) as rules.

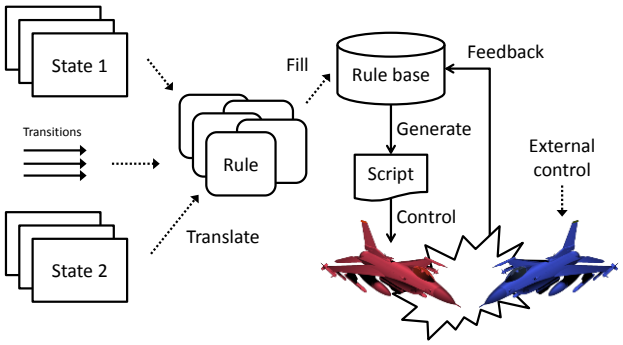


Figure 2. Adaptive finite-state machines through dynamic scripting.

of a CGF during an encounter with an opponent. At the end of the encounter (i.e., when one side is defeated and the simulation ends), a fitness value is calculated for the script, and this value is fed back to the rule base. The rule base updates the weights of the rules according to the fitness, in such a manner that rules that contributed to a high fitness value are rewarded with a weight increase, resulting in an increased probability of being selected the next time that a script is generated. Similarly, a low fitness results in a decrease of the weights of rules that contributed to this fitness value. The entire process of creating adaptive FSMs through dynamic scripting is illustrated in Figure 2. Through the use of behaviour rules, this process also enables the implementation of ethical decision-making. So far we have not concentrated on that topic, but we have set aside space in our technique for future implementations.

In the original description of dynamic scripting, rules are selected probabilistically, under the assumption that all rules are valid choices for inclusion in a script. However, for our goals these assumptions are invalid, as each state and each transition should be represented in a generated script. Not doing so could lead to scripts containing invalid FSMs. Two steps are required to resolve this issue. First, for a non-empty subset of states and transitions in the FSM, we create multiple interchangeable implementations, i.e., rules that trigger on the same conditions. These implementations express different but equally valid behaviours. In the case of states, each implementation provides behaviour that can be displayed in that state. In the case of transitions, each implementation provides a valid transition between states based on some conditions. Second, we alter the original dynamic scripting rule selection algorithm such that all states and transitions are represented in each script that

Algorithm 1. Script generation

Input: A rule base containing one or more implementations for each state and transition in a FSM.

Output: A script containing a rule for each state and transition in the FSM.

```
script = []
for element in fsm.get_elements():
   # fsm.get_elements() returns all states and
   # transitions in the FSM for which an
   # implementation needs to be included
   # in the script
  sum_of_weights = 0
  candidate_rules = []
  for rule in rule_base:
     # the rules in rule_base that are an
     # implementation of the current element are
     # added to a list of candidates for selection
    if rule.is_implementation_of(element):
      candidate_rules.append(rule)
      sum_of_weights += rule.weight
    end if
  end for
  if sum_of_weights == 0:
     # should the sum of the weights of the current
     # candidates be zero, we select a candidate at
     # random for inclusion in the script
    selected_rule = random.choice(candidate_rules)
    script.append(selected_rule)
  else:
     # we select a rule from candidate_rules through
     # roulette wheel selection based on the weights
     # of the candidate_rules
    selected_rule = roulette_wheel(candidate_rules)
    script.append(selected_rule)
  end if
end for
return script
```

is generated. This ensures that each generated script contains a completely valid FSM, and the proper set of rules concerning human values. This updated rule selection algorithm is shown in Algorithm 1.

As an example, consider the *Patrol* state from Figure 1a. One of the implementations of this state can be the rule definition as found in Figure 1b. An alternative implementation could be defined that directs the CGF aircraft to patrol in a triangular pattern a, b, and c rather than between points a and b. This implementation would be expressed by writing a new rule. Implementations of state transitions can be defined in a similar way, by using alternative preconditions for the rules governing the state transitions.

Air combat is a complex problem in a high-dimensional state space. Capturing air combat behaviour in rules greatly reduces the complexity of the generated behaviour models. The expert knowledge embedded in the rules enables the definition of behaviour for large parts of the state space, thereby quickly covering a large part of all possible situations. Furthermore, the dynamic scripting algorithm only recombines pieces of behaviour, and does not invent new pieces of behaviour. While this limits creativity, it also makes the behaviour generation system, and thereby the air combat task, easier to control and understand, compared to traditional machine learning methods. Finally, dynamic scripting is expected to converge quickly to satisfactory behaviour, as only a limited set of FSMs can be generated.

4 METHOD

To determine whether the method described in the previous section is capable of fast behaviour adaptation, we implemented the method in an air combat simulation using the STAGE [2] simulation environment. In this simulation, two cooperating CGFs, both controlled using an adaptive FSM, were tasked with the combat of two CGFs using static (non-adaptive) behaviour.

To be a suitable replacement for static CGFs, the adaptive CGFs should perform at least as well as the static CGFs. For this reason, we compare the performance of the adaptive CGFs to that of static CGFs using an FSM as currently found in Smart Bandits. Furthermore, to demonstrate the adaptive CGFs' adaptive capabilities, the adaptive CGFs will be placed in scenarios where they have to adapt from either an arbitrary initialization or after they have tuned their parameters to a previous opponent. These scenarios are analogous to generating good behaviour before any training by a human participant takes place, and adapting to changes in a human participant's behaviour during training.

The rest of this section describes the CGFs and the simulations in more detail.

4.1 CGFs

Both the static and adaptive teams consisted of two fighter jets (lead and wingman) equipped with radar, a radar warning receiver, and four semi-active long range missiles. The tactics used by the teams are described in Subsection 4.1.1 and 4.1.2.

4.1.1 Adaptive team

The FSMs used by the adaptive CGFs were based on an operational 2-versus-2 tactic. This tactic consists of two phases. The first phase is the opening sequence of the tactic, in which the CGFs detect the opposing CGFs, select an approach formation and assign targets between themselves. In the second phase, the CGFs engage and fire at their targets, after which they re-evaluate their tactical situation and either evade incoming missiles or select new targets.

For the adaptive CGFs, the tactic was subdivided into ten states in rule form. For this tactic, no meaningful new transitions could be identified, and as a result the original transitions were embedded in the rules created for the states. Next, new, additional implementations of selected states were designed and added as rules. Together with the original states and transitions, these rules formed the rule base for the adaptive CGFs. In total, 8 new states were added, resulting in a rule base with 18 rules. The adaptive lead and wingman were each assigned their own copy of the rule base, so that they could each optimize their own behaviour.

4.1.2 Static team

The scripts used by the static CGFs were based on one of two tactics. The first tactic (Tactic 1) was the same as the tactic used by the adaptive CGFs, resulting in a *mirror match*. By letting the adaptive CGFs fight against their own tactic, we will be able to show that they are able to improve on their own tactic using only a few extra variations of states. The second tactic (Tactic 2) was specifically designed to counter this tactic, to force the adaptive CGFs to come up with a creative solution.

Using these two tactics for the static team allows us to show different features of using the adaptive FSMs. By including the second tactic, we are able to show the adaptive capabilities of the adaptive CGFs, after they have already adapted to another tactic. This is in essence a form of transfer learning [19]. The ability to rapidly adapt to new tactics is important, as human trainees only spend a limited amount of time in a simulator, and ideally the adaptation of the adaptive CGFs is evident within that timeframe.

4.2 Learning parameters

We performed two types of simulations. First, the adaptive CGFs engaged the static CGFs using either Tactic 1 or Tactic 2 in fifty consecutive episodes, allowing the adaptive team to adapt to both tactics separately. In these cases, a baseline was set by engaging the static team with CGFs using the original (non-adaptive) Smart Bandits tactic. Second, the adaptive team, having already adapted to either Tactic 1 or Tactic 2, engaged the static team using the other tactic in fifty consecutive episodes. This demonstrates the "online" adaptivity of the adaptive CGFs. Each scenario was repeated ten times to obtain average performance data. For the baselines, each scenario was only repeated five times, as no learning took place.

Each trial ended when (1) a fighter jet on either side was hit with a missile[2], or (2) both sides had used all of their missiles, or (3) ten minutes of simulated time had passed. If an adaptive CGF had hit a static CGF, the adaptive team was declared the winner of the episode. In all other cases, the static team was declared the winner, even in a situation where no adaptive CGF was hit.

The dynamic scripting algorithm requires a fitness value as input, by which the proper weight adjustments are calculated. Earlier work determined the accumulated *probabilities-of-kill* of missiles fired to be effective fitness values for learning in the air combat domain [18]. However, we were unable to retrieve the necessary values to implement the probability-of-kill fitness from the STAGE API. Instead, a fitness of 1 was given to the winning team, and a fitness of 0 to the losing team. The weight adjustments were calculated as shown in Equation 1.

$$adjustment = max\ (\ -25,\ 50 * (\ (2 * fitness\ ) - 1\ )\ ) \qquad (1)$$

According to this Equation 1, the maximum possible reward is higher than the maximum possible punishment. This results in an algorithm that moves quicker into (local) optima than stepping back out of them.

5 RESULTS

We recorded which team successfully ended each episode, and calculated the win ratio as the number of wins divided by total number of repetitions of each episode. On average, each series of fifty episodes took 3.5 hours of real-time simulation.

Figure 3a shows the performance of the adaptive CGFs against the static CGFs using Tactic 1. The baseline CGFs fighting these static CGFs results in a mean win ratio of 0.46. The adaptive CGFs quickly converge to and hold a mean win ratio of 0.55, from episode 2 onwards. Optimal performance (0.80 mean win ratio) is first reached at episode 12, and again at episodes 37 and 41.

[2] Being outnumbered, the remaining team member is assumed to flee.

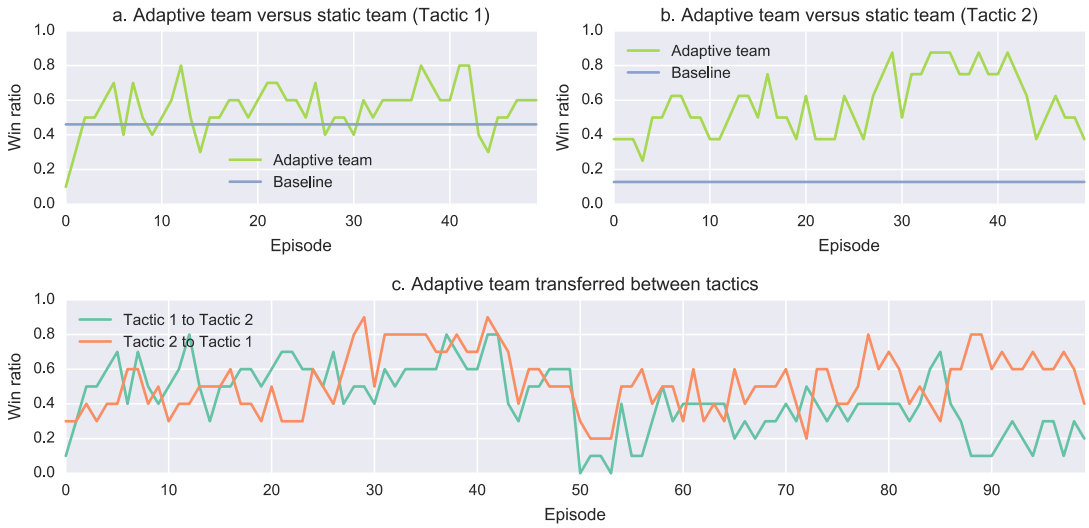


Figure 3. Performance of the adaptive CGFs against the static CGFs.

Figure 3b shows the same as Figure 3a, except for the static CGFs using Tactic 2. The baseline CGFs fighting these static CGFs results in a mean win ratio of 0.13. The adaptive CGFs' performance oscillates around 0.50 until episode 27. Between episodes 28 and 42, the performance spikes to a mean win ratio of 0.8, after which it drops again to the 0.50 level.

Figure 3c shows the performance of the adaptive CGFs when engaging the static CGFs, after the latter changed from Tactic 1 to Tactic 2 (green curve) and from Tactic 2 to Tactic 1 (orange curve). For the first 50 episodes, the same data is used that is also shown in Figures 3a and 3b. The remaining 50 episodes show the performance against the newly introduced tactics. The first peak reached in both cases are a mean win rate of 0.70 at episode 85 (Tactic 1 to Tactic 2), and 0.80 at episode 78 (Tactic 2 to Tactic 1).

6 DISCUSSION

The purpose of this study was to determine whether the method described in Section 3 is capable of fast adaptation of air combat behaviour. We tested this adaptive capability against static opponents that acted using two different tactics.

For the baselines, we relied on the performance of CGFs using the Smart Bandits tactic defined by experts. Figure Figure **3**a shows a win ratio near 0.50, which is expected as both sides repeatedly use the same tactics. However, random factors in the simulation environment (e.g., the hit rate of missiles) can still influence encounters.

Figures 3a and 3b show how well the adaptive CGFs are able to adapt to the two tactics employed by the static CGFs. Against Tactic 1, the adaptive CGFs are able to improve the baseline win ratio of 0.46, to a maximum of 0.80. This is a noteworthy result, as it shows that even with a limited amount of extra states, and given that the tactic taken from Smart Bandits was already optimized by experts, our algorithm was still able to further optimize the adaptive team's behaviour. During the design of scenarios, such a function may prove useful to aid the designers of opposing CGFs even before any training of human pilots takes place.

As mentioned in Subsection 4.1.2, the static CGFs' Tactic 2 was designed to defeat Tactic 1, which was employed in a non-adaptive manner in the baselines. The result is apparent in Figure 3b, with the baseline performance only reaching a 0.13 mean win ratio. The adaptive CGFs present a more positive picture. Although at first the performance stays around the 0.50 level, a new optimum is reached around episode 28. This optimum is maintained for about 15 episodes, after which the performance suddenly reverts to the old level. The high optimum indicates that the adaptive CGFs had good options (i.e., rules/states) to choose from, and the dynamic scripting algorithm was able to find the right combination quite efficiently.

The drop between episodes 40 and 45 signifies a certain brittleness of the system, as the adaptive CGFs are not able to hold their optimal solution. This is most likely caused by the random factors in the simulation environment, as mentioned earlier. A possible solution might be to introduce a memory of well-performing tactics, and to occasionally retry those tactics once the performance is dropping. Against static opponents, such memorized tactics could greatly mitigate the effect of random factors, and thereby increase the win ratio. Against other adaptive opponents (such as human trainees), retrying previously successful tactics may prove beneficial as well, especially if no other local optimum has been found for some time.

Of course, the adaptive CGFs had more options (i.e., pieces of behaviour) available to them than the baseline CGFs, meaning that the fact that they were able to defeat the static CGFs more often is not an impressive result by itself. However, what does matter is that the system can reach new performance levels, and maintain these levels for a significant amount of time. Furthermore, our system is able to let CGFs adapt their behaviour relatively fast, certainly when compared to systems employing creative methods such as neural networks and evolutionary algorithms.

An important use case for adaptive behaviour is online adaptation, i.e., adapting to the behaviour of human trainees during training. Figure 3c shows how well the adaptive CGFs can adapt to opponents using a new tactic, after having already adapted to earlier opponents with a different tactic. In both cases, a similar pattern is visible: the performance of the adaptive CGFs immediately dips when the new tactic is introduced, after which a moderate (0.40-0.60) performance level is held until a peak is reached around episode 80. With this kind of plasticity, the CGFs can quickly react to new tactics that human trainees may try out against them. Furthermore, with the low number of episodes needed to reach good behaviour (with e.g., a ≥0.5 win ratio), it

becomes feasible to run faster-than-real-time simulations between human-in-the-loop training sessions. This opens up the possibility of continuous adaptivity with a minimum amount of downtime, while keeping maximal control over the generated behaviour.

As mentioned in Section II, computational speed is crucial for machine learning in training simulations. Existing methods required large numbers of computational cycles to adapt CGF behaviour. Therefore, with regard to training simulations, the rapid adaptation as presented in this paper forms a substantial improvement over the existing methods. Field trials are currently underway with opponents controlled using adaptive FSMs flying against active RNLAF F-16 pilots. The results of these trials will be reported in the near future.

7 CONCLUSION

We have developed a machine learning method that is able to rapidly adapt the behaviour of CGFs to that of their opponents. The adaptive power of this method was shown in simulated air combat experiments. Compared to earlier work, the proposed method is computationally inexpensive and requires few iterations to generate good behaviour. Furthermore, the resulting behaviour models are in a format that is easily readable by human experts. This enables experts to effectively verify that the generated CGF behaviour complies with training goals and doctrine, including ethical decision-making. With adaptive CGFs as presented in this paper, military training simulations can be made more challenging and effective, leading to armed forces that are better prepared to defend shared values.

Future work includes evaluating the behaviour of the adaptive CGFs in human-in-the-loop trials, scaling up to engagements involving larger numbers of CGFs, and automatically setting up behaviour to realize predefined training goals in training simulations.

ACKNOWLEDGMENTS

The authors thank Remco Meiland, Joost van Oijen, Kees Krikke, Arnoud van Leeuwen, Nico Zink, and Aleid Duiker for their technical support and RNLAF Lt. Col. Roel Rijken for his continuous input.

REFERENCES

[1] J. D. Fletcher, "Education and training technology in the military," *Science (New York, N.Y.)*, vol. 323, pp. 72-75, January 2009.

[2] Presagis. (2015). *STAGE*. Available: http://www.presagis.com/files/product_brochures/2015-04-DS-SIM-STAGE14_web.pdf

[3] B. I. Simulations. (2014). *Virtual Battlespace 3*. Available: https://bisimulations.com/sites/default/files/BISim_VBS3_V2B_VBS IG_only_preview_2014%20%282%29.pdf

[4] R. Lopes and R. Bidarra, "Adaptivity challenges in games and simulations: a survey," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 3, pp. 85-99, 2011.

[5] N. Abdellaoui, A. Taylor, and G. Parkinson, "Comparative Analysis of Computer Generated Forces' Artificial Intelligence," in *RTO-MP-MSG-069 - Current uses of M&S Covering Support to Operations, Human Behaviour Representation, Irregular Warfare, Defence against Terrorism and Coalition Tactical Force Integration*, Brussels, Belgium, 2009.

[6] NLR. (2016). *Smart Bandits AIR*. Available: http://www.nlr.org/capabilities/smart-bandits-air/

[7] A. Toubman, G. Poppinga, J. J. Roessingh, M. Hou, L. Luotsinen, R. A. Løvlid, *et al.*, "Modeling CGF Behavior with Machine Learning Techniques: Requirements and Future Directions," in *Proceedings of the 2015 Interservice/Industry Training, Simulation, and Education Conference*, Orlando, Florida, 2015, pp. 2637-2647.

[8] R. L. Shaw, *Fighter Combat*: Naval Institute Press, 1985.

[9] E. Y. Rodin and S. M. Amin, "Maneuver prediction in air combat via artificial neural networks," *Computers & mathematics with applications*, vol. 24, pp. 95-112, 1992.

[10] T.-H. Teng, A.-H. Tan, and L.-N. Teow, "Adaptive computer-generated forces for simulator-based training," *Expert Systems with Applications*, vol. 40, pp. 7341-7353, 2013.

[11] T.-H. Teng, A.-H. Tan, Y.-S. Tan, and A. Yeo, "Self-organizing neural networks for learning air combat maneuvers," in *Neural Networks (IJCNN), The 2012 International Joint Conference on*, 2012, pp. 1-8.

[12] S. Mulgund, K. Harper, K. Krishnakumar, and G. Zacharias, "Air combat tactics optimization using stochastic genetic algorithms," in *Proceedings of the 1998 IEEE International Conference on Systems, Man and Cybernetics*, 1998, pp. 3136-3141.

[13] K. Zhang, W. Li, and Z. Wang, "Large-Scale Air-Combat Formation Optimization Using Simulated-Annealing GA (Genetic Algorithm)," in *Proceedings of the 24th Congress of International Council of the Aeronautical Sciences*, Yokohama, Japan, 2004.

[14] R. E. Smith, B. A. Dike, B. Ravichandran, A. El-Fallah, and R. K. Mehra, "The fighter aircraft LCS: A case of different LCS goals and techniques," in *Learning Classifier Systems*, ed: Springer, 2000, pp. 283-300.

[15] R. E. Smith, B. A. Dike, R. K. Mehra, B. Ravichandran, and A. El-Fallah, "Classifier systems in combat: two-sided learning of maneuvers for advanced fighter aircraft," *Computer Methods in Applied Mechanics and Engineering*, vol. 186, pp. 421 - 437, 2000.

[16] J. Yao, Q. Huang, and W. Wang, "Adaptive Human Behavior Modeling for Air Combat Simulation," in *2015 IEEE/ACM 19th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, 2015, pp. 100-103.

[17] R. Kop, A. Toubman, M. Hoogendoorn, and J. Roessingh, "Evolutionary Dynamic Scripting: Adaptation of Expert Rule Bases for Serious Games," in *Current Approaches in Applied Artificial Intelligence*. vol. 9101, M. Ali, Y. S. Kwon, C.-H. Lee, J. Kim, and Y. Kim, Eds., ed: Springer International Publishing, 2015, pp. 53-62.

[18] A. Toubman, J. J. Roessingh, P. Spronck, A. Plaat, and J. Van den Herik, "Rewarding Air Combat Behavior in Training Simulations," in *Proceedings of the 2015 IEEE International Conference on Systems, Man and Cybernetics*, Hong Kong, 2015, pp. 1379-1402.

[19] A. Toubman, J. J. Roessingh, P. Spronck, A. Plaat, and J. Van den Herik, "Transfer Learning of Air Combat Behavior," in *Proceedings of the 2015 IEEE International Conference on Machine Learning and Applications*, Miami, Florida, 2015.

[20] P. Spronck, M. Ponsen, I. Sprinkhuizen-Kuyper, and E. Postma, "Adaptive game AI with dynamic scripting," *Machine Learning*, vol. 63, pp. 217-248, 2006.

[21] D. Fu and R. Houlette, "The Ultimate Guide to FSMs in Games," in *AI Game Programming Wisdom*. vol. 2, S. Rabin, Ed., ed: Charles River Media, 2004, pp. 283-302.

[22] J. J. Roessingh, R.-J. Merk, P. Huibers, R. Meiland, and R. Rijken, "Smart Bandits in air-to-air combat training: Combining different behavioural models in a common architecture," in *21st Annual Conference on Behavior Representation in Modeling and Simulation*, Amelia Island, Florida, USA, 2012.

[23] A. Khatami, P. Huibers, and J. J. Roessingh, "Architecture for goal-driven behavior of virtual opponents in fighter pilot combat training," in *Proceedings of the 22nd Annual Conference on Behavior Representation in Modeling and Simulation*, Ottawa, Canada, 2013.