Improving Air-to-Air Combat Behavior Through Transparent Machine Learning

Armon Toubman, Jan Joris Roessingh
National Aerospace Laboratory NLR
Amsterdam, the Netherlands
{Armon.Toubman, Jan.Joris.Roessingh}@nlr.nl

Pieter Spronck, Aske Plaat
Tilburg University
Tilburg, the Netherlands
{p.spronck, aske.plaat}@gmail.com

Jaap van den Herik
Leiden University
Leiden, the Netherlands
jaapvandenherik@gmail.com

ABSTRACT

Training simulations, especially those for tactical training, require properly behaving computer generated forces (CGFs) in the opponent role for an effective training experience. Traditionally, the behavior of such CGFs is controlled through scripts. There are two main problems with the use of scripts for controlling the behavior of CGFs: (1) building an effective script requires expert knowledge, which is costly, and (2) costs further increase with the number of 'learning events' in a scenario (e.g. a new opponent tactic). Machine learning techniques may offer a solution to these two problems, by automatically generating, evaluating and improving CGF behavior. In this paper we describe an application of the dynamic scripting technique to the generation of CGF behavior for training simulations. Dynamic scripting is a machine learning technique that searches for effective scripts by combining rules from a rule base with predefined behavior rules. Although dynamic scripting was initially developed for artificial intelligence (AI) in commercial video games, its computational and functional qualities are also desirable in military training simulations. Among other qualities, dynamic scripting generates behavior in a transparent manner. Also, dynamic scripting's learning method is robust: a minimum level of effectiveness is guaranteed through the use of domain knowledge in the initial rule base. In our research, we investigate the application of dynamic scripting for generating behaviors of multiple cooperating aircraft in air-to-air combat. Coordination in multi-agent systems remains a non-trivial problem. We enabled explicit team coordination through communication between team members. This coordination method was tested in an air combat simulation experiment, and compared against a baseline that consisted of a similar dynamic scripting setup, without explicit coordination. In terms of combat performance, the team using the explicit team coordination was 20% more effective than the baseline. Finally, the paper will discuss the application of dynamic scripting in a practical setting.

ABOUT THE AUTHORS

Armon Toubman is a Ph.D. student at the National Aerospace Laboratory and Tilburg University in the Netherlands. His research area is the use of machine learning to automatically generate behavior for computer generated forces in air-to-air combat simulations. He holds a Master of Science in Artificial Intelligence from VU University Amsterdam.

Jan Joris Roessingh holds a Ph.D. in Physics from Utrecht University. Since 1994 he has worked as a scientist at the department of Training, Simulation and Operator Performance of the National Aerospace Laboratory in Amsterdam, the Netherlands. His research interests focus on operational improvements in aerospace applications and development of training simulations as well as the modeling and measuring of patterns of skilled and learning behavior, both in individual and team settings.

Pieter Spronck holds a Ph.D. in Artificial Intelligence from Maastricht University. He currently works as an associate professor at the Tilburg center for Creative Computing. His research focuses on computer game AI and multi-agent systems.

Aske Plaat is currently professor of Information and Complex Decision Making at Tilburg University and associate professor of Data-driven Entertainment at NHTV University of Applied Sciences Breda. His main fields of research are data science, machine learning, and artificial intelligence. He holds a Ph.D. in Artificial Intelligence from Erasmus University Rotterdam.

Jaap van den Herik holds a position as professor at the Leiden Institute for Advanced Computer Science, and as professor of Law and Computer Science at the Faculty of Law of Leiden University. His current research focus is on big data technology. He holds a Ph.D. in Artificial Intelligence from Delft University of Technology.

Improving Air-to-Air Combat Behavior Through Transparent Machine Learning

Armon Toubman, Jan Joris Roessingh
National Aerospace Laboratory NLR
Amsterdam, the Netherlands
{Armon.Toubman, Jan.Joris.Roessingh}@nlr.nl

Pieter Spronck, Aske Plaat
Tilburg University
Tilburg, the Netherlands
{p.spronck, aske.plaat}@gmail.com

Jaap van den Herik
Leiden University
Leiden, the Netherlands
jaapvandenherik@gmail.com

INTRODUCTION

Simulation has become a mainstay in many fields (Bair & Jackson, 2013). In the field of military training, simulation is an invaluable tool. Real-life exercises are expensive, dangerous and time-consuming to set up, while simulations are relatively cheap, safe, and flexible (Fletcher, 2009; Laird, 2000).

In military simulations, the roles of allies and opponents are often performed by computer generated forces (CGFs). In high-fidelity simulators, the fidelity of the behavior of these CGFs is crucial to the overall training experience. Traditionally, the behavior of CGFs is scripted (Roessingh, Merk, Huibers, Meiland, & Rijken, 2012). Production rules—rules that map conditions to actions—are manually crafted to suit specific (types of) CGFs. In complex domains such as air combat, scripts for CGFs rapidly become complex and require substantial domain expertise to create.

Artificial Intelligence techniques may provide a solution to the problem of generating behavior for CGFs. Many different approaches have already been attempted. The use of cognitive models is one seemingly popular approach, which can be found in well-known systems such as TacAir-Soar (Jones et al., 1999) and ACT-R (Anderson, 1996). At the National Aerospace Laboratory (NLR), recent work on CGF behavior has focused on optimizing cognitive models with machine learning techniques such as neural networks and evolutionary learning (Koopmanschap, Hoogendoorn, & Roessingh, 2013). However, such methods result in large, opaque behavior models which are hard to understand and reuse.

In this paper, we return to scripts for the transparency of production rules. To ease the difficulty of composing scripts that should result in effective behavior based on the rules given, we apply a machine learning technique called dynamic scripting (DS) (Spronck, Ponsen, Sprinkhuizen-Kuyper, & Postma, 2006). DS was originally developed for the generation of behavior for AI characters in video games and was designed with certain functional and computational characteristics in mind that transfer well to the domain of military training.

We extended DS with a method for team coordination, which we call DS+C. This extension enables CGFs to automatically learn to coordinate actions through communication. At the end of the learning process, the resultant coordination between the CGFs is understandable and transparent. This transparency is a direct result of the fact that DS requires premade rules to learn from and outputs combinations of these rules. We demonstrate both the machine learning process and the DS+C extension with an air combat simulation experiment. The results of this experiment show that the use of DS+C leads to a 20% performance increase over the use of DS without team coordination.

This work is targeted to practical military applications and, as such, contains new material that builds forth on previous work (Toubman, Roessingh, Spronck, Plaat, & Herik, 2013).

DYNAMIC SCRIPTING AND RELATED WORK

Dynamic scripting is an automated learning technique based on reinforcement learning, introduced by Spronck (Spronck et al., 2006). In essence, DS uses a weighted rule selection mechanism to select rules from a rule base and generate a script. The generated script governs the behavior of a simulated agent during an encounter with some opponent agent or agents.

The rule base is initialized with predefined behavior rules. These rules are pieces of behavior based on domain knowledge. In other words, the rule base should be initialized with rules that typify behavior that is possibly useful to the agent in situations that this agent might encounter.

The DS algorithm assigns a constant initial weight to each rule in the rule base, and begins a selection process. A preset number of rules are randomly selected from the rule base according to their weights. The selected rules form a script, which is used to control an agent during an encounter. The result of this encounter is fed back to the DS algorithm which calculates a fitness score based on this result. The calculated fitness score is used to update the weights of the rules that were activated during the encounter, either positively or negatively. The change in weight affects the likelihood of the rules to be selected for a new script. This process is repeated until some goal is reached. For example, in the air combat domain, such a goal might be reaching a point at which scripts are generated that can reliably defeat an opposing force. The DS learning process is illustrated in Figure 1. See Spronck et al. (2006) for full details on the DS algorithm.

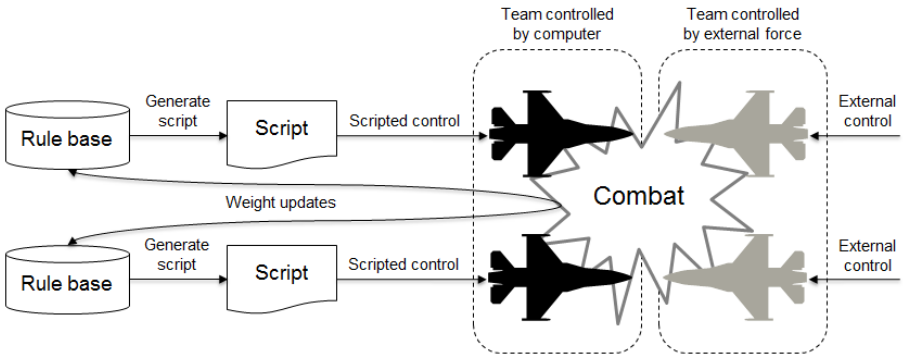

Figure 1. Dynamic scripting in the context of two learning agents in an air combat simulation.

Because of the stochastic nature of the rule selection mechanism, it is important to evaluate samples of generated scripts. To this end, Spronck defined a measure called the turning point, which is the encounter after which the learning agents reached a higher fitness value than the opposing agents for at least ten consecutive encounters. At the turning point, it can be said that the learning agents are consistently outperforming their opponents.

DS was intended as a method for automated generation of behavior for AI characters in commercial video games. In the video game industry, automated generation of AI behavior is rarely used because of the possibility of unwanted behavior emerging, which lowers the overall quality of the game and leads to negative reviews. DS was designed with several computational and functional requirements in mind to address this issue. These requirements are listed as speed (the learning algorithm should be fast), effectiveness (all behavior should be at least reasonably effective in some situations), robustness (results are sampled and extreme results do not lead to extreme weight updates), clarity (scripts are easily understood by humans), and variety (the selection mechanism ensures behavior will be generated in different combinations). The same requirements are applicable in the field of military training simulations, where the quality of the behavior of constructive agents is as important as it is in commercial video games.

In practice, DS can be applied in both an online and offline fashion. In other words, DS can be used to learn initial behavior in an automated way (i.e., offline learning). Then, when the weights of the rules have sufficiently converged, the algorithm can be left active when the learning agents are set up against human trainees. When the

result of such an encounter is fed back to DS, the weights of the rules will again be updated. This way, while the training simulations are underway, the agents will still be able to learn and try different scripts against different strategies that the humans might be employing. Alternatively, the learning process can be stopped and the resulting rules and their weights can be inspected. Static scripts can then be manually extracted and possibly tweaked.

When DS is used to control a team of agents (by assigning each agent its own rule base and DS instance, i.e., decentralized control), team behavior is the result of emergence. However, it may be desirable and even advantageous to have the agents in a team display some form of team coordination. Especially in military training simulations, agents have to be able to coordinate movements like staying in formation and performing tactical maneuvers.

In general, there are two methods of team coordination: centralized and decentralized coordination (van der Sterren, 2002). With centralized coordination, one agent coordinates the actions of multiple agents. With decentralized coordination, all agents in a team may influence each other's actions by sharing information through some form of communication.

For this paper, we have chosen to implement decentralized team coordination. With DS, decentralized control translates to multiple agents having their own rule base and instance of the learning algorithm. We achieved coordination between the agents through communication. Adding communication to multi-agent systems is not a trivial problem (Stone & Veloso, 2000). For this reason, we attempted to fit the communication scheme (and therefore also the coordination) into the learning mechanism of DS.

DYNAMIC SCRIPTING WITH TEAM COORDINATION

We implemented team coordination with DS through communication between agents resulting in a technique we call DS+C. By utilizing the production rules for communication by sending messages, and letting these messages trigger behavior in the recipients, the DS algorithm is able to learn which exchanges of messages result in the most effective team behavior.

In general, the communication scheme consists of three parts. First, one additional action is added to each rule of each agent. This action is to send a message to team members containing the intention of that rule. For example, rules for agents in an air combat simulation might be described as 'evasive' or 'aggressive'.

The second part of the communication scheme is a new component for the agents. This component stores messages that are received, until the agent has processed its rules. This component is needed to account for any asynchronous processing between the agents.

The third part is the addition of new rules to the rule bases of the agents. These rules, (i.e., the 'coordination rules'), are designed in such a way that the reception of messages containing intentions of other agents trigger some form of corresponding behavior. The complete communication scheme is shown in Figure 2.

Using this method, we obtain agents with rule bases that contain rules that are proactive (act and send messages) and reactive (act on received messages) regarding team coordination. The rule bases can also contain variants of the rules that send messages or act on received messages. Because of the way the DS learning algorithm works, the rules are recombined into scripts, and tested in simulated encounters. This way, the agents are able to learn which exchanges of messages lead to the most effective behavior.

The specificity of the intentions that are sent in the messages should be tailored to the application. If the intentions are described too narrowly, it is possible that DS will not be able to match up the proactive rules of one agent with the right reactive rules in another agent. However, if the intentions are described too widely, the chance of unrelated behavior being coordinated between agents increases.

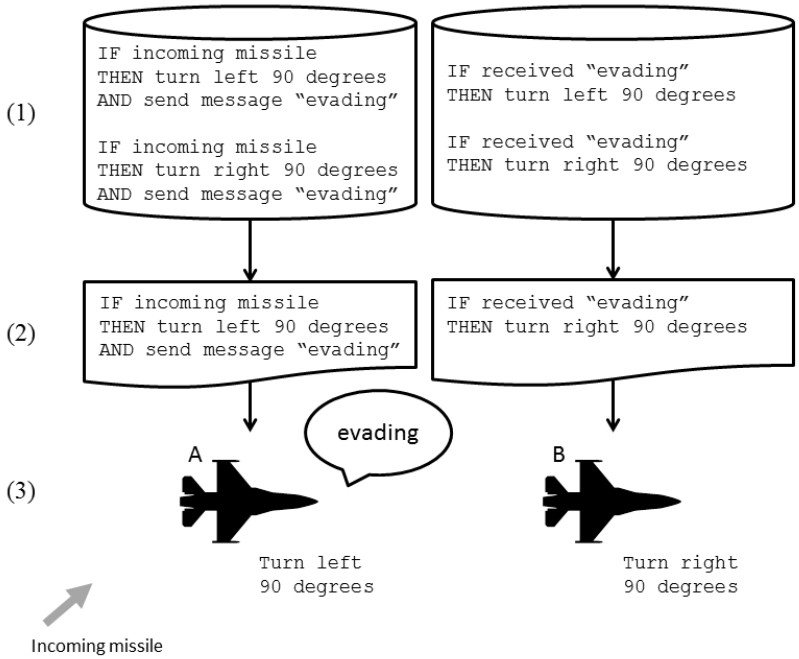

Figure 2. Example of DS+C in action. (1) Proactive and reactive coordination rules are added to the rule bases. The left rule base shows two example proactive rules, the right rule base shows to example reactive rules. Variations on the same rules are made to give the DS algorithm more options to explore. (2) The DS algorithm selects rules from the rule bases to form scripts (in this case, one rule per script). (3) Agent A's rule is activated by an incoming missile. A turns, and sends a message "evading" to B. This activates B's rule, causing B to turn.

CASE STUDY

As an exploratory study, we tested the application of DS+C in a custom air-to-air combat simulation. In the simulation, a formation of two blue fighters ("the blues"), a lead and a wingman, had to eliminate a single red fighter. The red fighter flew a Combat Air Patrol (see Figure 3) to defend an area of airspace. The mission of the blues was considered successful if they eliminated the red fighter without any losses on their own side. The mission of red was to eliminate all fighters it detected. Figure 4 shows a screenshot of the simulation.

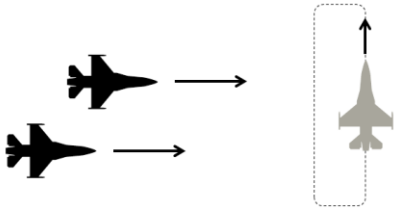

Figure 3. The scenario used in the case study. The blue fighters (left) fly towards the red fighter (right). The red fighter is flying a CAP.

The behavior of the blues was governed by scripts generated using the DS+C method. The rules used by the blues are divisible into roughly three sets.

The first set consists of default rules. These are rules that define basic behavior that is needed in every encounter, and on which the agents can fall back if no other rules apply. These rules are included in every script, and their weights are left unchanged by the DS+C process. An example of a default rule is to fly in the direction of the airspace that red is patrolling if no other rule applies.

The second set consists of general rules for air combat. These rules define behavior such as tracking enemies on the radar, firing missiles at enemies and evading incoming missiles. These rules are based on domain knowledge, although highly simplified to illustrate the

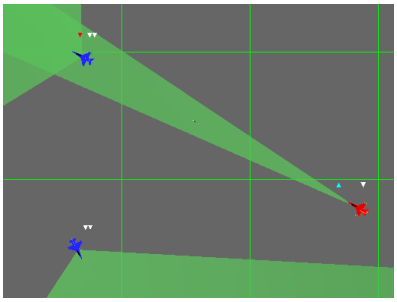

Figure 4. Screenshot of the simulation.

principles. Also, these are the rules that send the intention of the agent as a message (i.e., the proactive rules).

The third set consists of the coordination rules. These are the reactive rules that activate in response to the reception of messages from other agents (or in our case, just the other blue agent).

DS+C generated scripts with six rules plus the default rules. Each rule base had 31 rules that started with a weight of 50. Each rule was also manually assigned a priority number. In case multiple rules activated at the same time, this priority number would be used to determine precedence. In case the priority numbers were tied, the rule with the highest weight would be chosen. A sample of four rules is shown in Table 1.

Table 1. A sample of the rules that the blues used in the experiment.

Rule	Priority
If my teammate is alive, then fly in a '2-ship element' formation	1
If my Radar Warning Receiver detects an enemy radar, then turn approximately towards that radar.	6
If my radar is in 'lock' mode and I have missiles left and I am within 80 units from the enemy, then fire a missile.	9
If I receive a message that my teammate is evading the enemy's radar, change my heading 90 degrees plus the approximated relative bearing to the nearest enemy radar that is detected by my Radar Warning Receiver.	6

The red fighter used three basic tactics which were implemented as three static scripts. With the *Default* tactic, red flew a basic CAP and engaged all enemies it detected, as described earlier. Red also used an *Evading* tactic with which it would try to evade the radar of the blues, and a *Close Range* tactic with which it would only fire missiles if the blues were in close range (that is, closer than with the *Default* tactic). These three basic tactics each had an alternative version in which red would fly its CAP in clockwise direction rather than in counter-clockwise direction, to introduce more variety in the encounters. Finally, to test how well the blues would be able to learn generalized behavior, red was given a composite tactic which consisted of the three basic tactics plus their alternative versions. With these *mixed tactics*, red randomly selected a tactic and would use this tactic until it lost an encounter, at which point it would randomly select a new tactic to use.

DS uses a fitness function to measure the performance of agents in an encounter. We measured the performance of the blues using the following fitness function:

$$fitness = \big(0.25 + (0.5 * winner)\big) + 0.125 * speed + 0.125 * resources \quad (1)$$

In Equation 1, *winner* is 1 if the blues won or 0 if they lost; *speed* is 1 minus the ratio of the duration of the encounter to the maximum allowed duration; and *resources* is a value between 0 and 1 based on the number of missiles spent in the trial (stimulating the blues to eliminate red with as few missiles as possible). The fitness function is used to calculate the weight adjustments between trials. Unfortunately there is no standard way to do this, as the calculations have to be tailored to the output of the fitness function. We used the following function:

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

The constants in Equation 2 represent the balance between reward and punishment. For example, we set the maximum negative weight adjustment to -25, such that rules that started at the initial weight of 50 would still have a reasonable chance to be selected in a subsequent trial.

Below, we compare the performance of DS+C to that of regular DS. In order to do so, we first define performance in terms of efficiency (learning speed) and effectiveness (combat results). We define effectiveness as the mean win/loss ratio during a learning episode. It is difficult to define the efficiency of the DS algorithm, because it is hard to establish precisely when stationary performance is reached. Both the DS algorithm and the simulated environment are stochastic by nature. For this reason, it is unlikely that the DS algorithm will converge to a single optimal script. A performance measure is needed that takes this fact into account.

To deal with the inherent variations in the learning process, we define the turning point measure TP(x) (based on Spronck's TP measure (Spronck et al., 2006)) as the encounter after which the blues have won x percent of the last 20 encounters. The window size of 20 encounters was chosen to allow for a sufficient number of evaluation points during a learning episode. A learning episode consisted of 250 encounters. The x thus represents the chance that at TP(x) a winning script will be generated. It now follows that an early TP(x) represents a more efficient learning process, while a late TP(x) represents a less efficient learning process.

Two sets of experiments were run. In the first set, the blues used regular DS, while in the second set they used DS+C. Red used one of the seven tactics described earlier (see Table 2). The results of the experiments are described in the next section.

RESULTS

For each of the basic tactics used by red, results were averaged over 10 learning episodes, with each episode consisting of 250 trials. In the case of the mixed tactics, the results were averaged over 100 learning episodes (to reduce noise in the data).

The average TP(x) was calculated with x being 50%, 60%, 70% and 80% wins, for the performance of the blues against each of red's tactics. These specific values for x were chosen because they represent the most interesting ranges: win/loss ratios below 50% mean a majority of losses, while win/loss ratios over 80% are unlikely due to the stochastic nature of DS. For the mixed tactics, the TPs were compared using independent two-sample t-tests. Learning curves were created using a rolling average of the win/loss ratio, with a window of 20 encounters. Also, a log of the weight changes was kept to be able to see to what extent the coordination rules were selected by the DS+C agents.

Table 2 shows the TPs of the blues using DS and DS+C against red's individual basic tactics and mixed tactics. Independent two-sample two-tailed t-tests show that against the mixed tactics at TP(50%), the mean TPs are achieved significantly earlier using DS+C (t = 3.85, p = 0.00016) at the a = 0.05 significance level. The same holds for TP(60%) (t = 3.60, p = 0.00039), TP(70%) (t = 3.60, p = 0.00039), and TP(80%) (t = 2.46, p = 0.015).

Figure 4 shows the learning curves of the agents with DS and DS+C against red's mixed tactics. After around 100 trials, both the DS and DS+C agents seem to reach a plateau. After the first 100 encounters, DS and DS+C maintain a mean win/loss ratio of respectively 0.53 and 0.63. During the entire learning process, the DS+C agents clearly outperformed the DS agents, with a mean difference in win/loss ratio of 20.3%.

We found in the simulation logs that the coordination rules were selected and activated multiple times. Several rules out of the 31 rules each blue had in its rule base received particularly high weights. The blue lead favored one rule in particular, with a mean final weight of 178.6. This rule stated 'if I receive a message that my wingman is evading an enemy, turn approximately towards that enemy'. The blue wingman mainly favored two rules that made it perform evasive actions when it received a message from the lead that it was trying to avoid being detected by red, with mean final weights of 103.8 and 106.6.

The rule that was favored the most in the case without coordination was the 'beam maneuver' (flying perpendicular to an enemy's radar to avoid detection). This rule received high weights from both the blue lead (386.7) and the blue wingman (323.5). Also, in all cases, the blues preferred firing missiles from a greater distance.

Table 2. TPs of DS and DS+C against red's basic tactics (aggregated results, 10 learning episodes per tactic) and the mixed tactics (averaged over 100 learning episodes).

		TP(50%)		TP(60%)		TP(70%)		TP(80%)	
Tactics of red	**DS**	**μ**	**σ**	**μ**	**σ**	**μ**	**σ**	**μ**	**σ**
Patrol	DS	30	13.9	33.1	14.3	48.1	20.6	108	74.3
	DS+C	23.4	5.3	25.8	7.2	28.1	8.4	31.2	11.1
Patrol (Evading)	DS	49.4	17.3	67.7	31	78.3	28	108.9	59.5
	DS+C	28.1	8.1	34.8	10.4	52.4	38.6	77.9	77.9
Patrol (Close Range)	DS	22.4	3.4	24.2	5.9	29.6	12	33.1	13.7
	DS+C	25.7	8.9	35.9	17.7	58	61.2	63.6	68.7
Patrol (alt.)	DS	42.3	33.1	62.5	42.1	90.9	43.7	114.1	64.7
	DS+C	31.1	10.5	35.3	10.2	43.8	12.7	73.8	65.3
Patrol (Evading, alt.)	DS	131.3	76.4	137.5	74.5	156	82.7	200.9	58.2
	DS+C	54.4	70.7	66.4	67.8	91.3	86.7	143.6	92.7
Patrol (Close Range, alt.)	DS	59.1	71	71.6	67.3	121.2	82.8	169.4	92.5
	DS+C	45.9	17.1	90	57.9	116.4	75.6	154.2	77.5
Mixed	DS	83.8	78.1	94.5	78.9	110.5	78.4	129.9	79.1
	DS+C	48.4	48.4	60.9	49.6	75.8	55.5	103.9	69.7
Basic (average)	DS	55.8	56.7	66.1	57.8	87.3	66.5	122.4	82
	DS+C	34.8	31.3	48	42.7	65	61.3	90.7	80.6

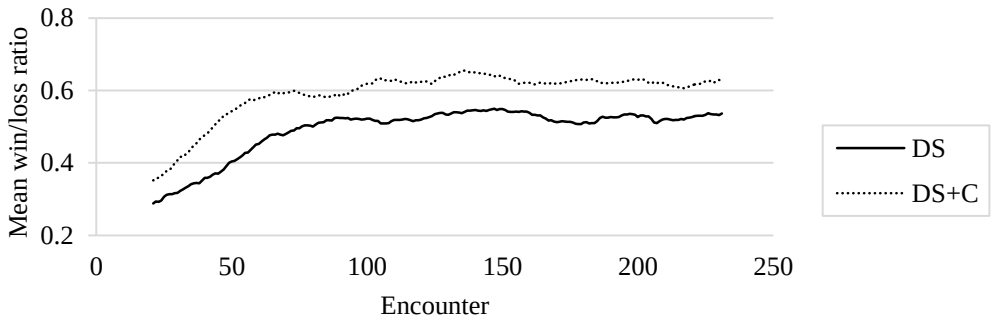

Figure 5. Rolling average (window size 20) of win/loss ratio of the blues against red's mixed tactics, with DS and DS+C. Ratios are averaged over 100 learning episodes.

DISCUSSION

Over a large set of experiments, DS+C showed clear advantages over traditional DS for multi-agent reinforcement learning. Throughout our experiments, the DS+C agents won more often than DS agents from an opponent that frequently changed its tactics. The DS+C agents also improved on the TPs set by the regular DS agents in encounters against both a predictable (basic tactics) and unpredictable (mixed tactics) agent. Coordination in multi-agent systems is an extensively researched topic with many open issues and learning opportunities (Stone & Veloso, 2000). The literature shows that the addition of coordination to a multi-agent system does not automatically lead to increased performance (Balch & Arkin, 1994), which makes the results obtained with DS+C especially noteworthy.

We used the newly defined TP(x) measure to compare the efficiency of DS and DS+C. Against an opponent with mixed tactics, the DS+C agents reached the TP(x) at 50%, 60%, 70% and 80% significantly earlier than the agents with regular DS did. Similar patterns can be seen in the results for the basic tactics. In other words, in our experiments, these 'milestones in learning' were reached earlier by DS+C. Therefore, we may provisionally conclude that DS+C agents learn more efficiently than DS agents.

The learning curves in Figure 5 show that the DS+C achieved and maintained a higher average win/loss ratio throughout the learning process, against an opponent using mixed tactics. Therefore, we provisionally conclude that apart from being more efficient, DS+C agents are also more effective than DS agents, both during and after the learning process.

We mainly attribute the higher performance of the DS+C agents to the higher count of rules leading to evasive actions in the rule bases. The blues lost an encounter if a single blue was hit by a missile from red.[1] Therefore cautious behavior was rewarded. This can be seen in the high weights that several evasion rules received. This also possibly explains the earlier convergence to optimal scripts, since the DS+C agents had more good options available. However, the coordination rules were not intentionally biased towards evasive actions, and it is possible that more aggressive rules would have a similar effect. At the same time, the fact that the blues together had more missiles at their disposal than red is likely to also have contributed to the emergence of a low-risk strategy.

The results show a slightly different picture against the opponent using each of the basic tactics. As Table 2 shows, DS+C also reached the TPs earlier against the basic tactics on average, but with a smaller lead. Only against the *Close Range* tactic did DS achieve the TPs earlier. We hypothesize that if DS was able to rapidly find optimal behavior against this tactic of 'red', then the additional coordination rules for DS+C only hindered the convergence to successful rules, resulting in later TPs.

Furthermore, we detected a trend in the results of both DS and DS+C having relatively late TPs against the alternative versions (with reversed direction) of red's tactics. Additional experiments, in which the formation of the blues was mirrored, also led to later TPs. This could be considered an artefact in the experiments, or it could be an indication that the exact spatial configuration of cooperating aircraft is a relevant factor in air combat.

Overall, we find that DS+C is an extension that has the ability to improve both the efficiency and effectiveness of DS. The results found here could be improved even further with a better understanding of the interactions and the effects of the rules that were used.

CONCLUSIONS AND FUTURE WORK

We have presented a method for team coordination through communication using DS, called DS+C. DS is a transparent machine learning method, which combines predefined rules to produce behavior for agents. Our DS+C extension provides team coordination using the same transparent learning mechanism. The main benefit of DS+C over DS that we have found was to better generalize learned behavior against unpredictable opponents. This way, the agents will be better suited for use in training simulations involving human participants who might act unpredictably as well.

We would like to emphasize that DS was originally developed for use in commercial video games. Since video games and training simulations share many requirements regarding behavioral realism and quality control, we advise to maintain a strong link between these fields.

In practical terms, we find that DS can be used in three different ways. First and foremost, it can be used to rapidly generate behavior for CGFs. This can be done in an offline fashion by setting up the learning agents against other AI enemies. Second, DS can be used to keep the behavior of the agents adaptive in encounters against humans. By keeping the learning mechanism active during the actual training sessions, the DS agents will continue to learn and adapt to the behavior of the human trainees. Third, DS provides a transparent test bed for research on agent behavior. This is demonstrated in how we used the DS algorithm to determine which exchanges of messages provided the most benefit to our learning agents. The same setup could be used for similar behavior extensions.

Future work will focus on the use of only one DS instance to generate behavior for multiple agents. The reduced complexity might have an even greater positive impact on learning speeds. Furthermore, the design and application

[1] The possibility of partial wins—situations where red would shoot down one blue, but the surviving blue would still destroy red—was considered, but a correct weight update for both blues was hard to achieve due to the interactivity of the rules of the blues.

of more elaborate pre-scripted tactics using DS will be investigated, with the goal of adding more realistic behavior options.

ACKNOWLEDGEMENTS

The authors thank LtCol Roel Rijken (Royal Netherlands Air Force) for the first version of the simulation environment and advice regarding air combat tactics, Pieter Huibers for further work on the simulation and personal assistance, and Xander Wilcke for even further work on the simulation.

REFERENCES

Anderson, J. (1996). ACT: A simple theory of complex cognition. *American Psychologist*, *Vol 51*(4), 355–365. doi:10.1037/0003-066X.51.4.355

Bair, L., & Jackson, J. (2013). M&S Professionals Domains, Skills, Knowledge, and Applications. *The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC)*, 1435–1445. Retrieved from http://ntsa.metapress.com/index/X0M222V306264583.pdf

Balch, T., & Arkin, R. C. (1994). Communication in reactive multiagent robotic systems. *Autonomous Robots*, *1*(1), 27–52. doi:10.1007/BF00735341

Fletcher, J. D. (2009). Education and training technology in the military. *Science (New York, N.Y.)*, *323*(5910), 72–5. doi:10.1126/science.1167778

Jones, R. M., Laird, J. E., Nielsen, P. E., Coulter, K. J., Kenny, P., & Koss, F. V. (1999). Automated intelligent pilots for combat flight simulation. *AI Magazine*, *20*(1), 27–42. Retrieved from http://www.aaai.org/ojs/index.php/aimagazine/article/viewArticle/1438

Koopmanschap, R., Hoogendoorn, M., & Roessingh, J. J. (2013). Learning Parameters for a Cognitive Model on Situation Awareness. In *The 26th International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems* (pp. 22–32). Amsterdam, the Netherlands.

Laird, J. E. (2000). An exploration into computer games and computer generated forces. In *Eighth Conference on Computer Generated Forces and Behavior Representation*.

Roessingh, J. J., Merk, R.-J., Huibers, P., Meiland, R., & Rijken, R. (2012). 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.

Spronck, P., Ponsen, M., Sprinkhuizen-Kuyper, I., & Postma, E. (2006). Adaptive game AI with dynamic scripting. *Machine Learning*, *63*(3), 217–248. doi:10.1007/s10994-006-6205-6

Stone, P., & Veloso, M. (2000). Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, *8*(3), 345–383.

Toubman, A., Roessingh, J. J., Spronck, P., Plaat, A., & Herik, J. van den. (2013). *Dynamic Scripting with Team Coordination in Air Combat Simulation (TP-2013-507)*. Amsterdam, the Netherlands. Retrieved from http://reports.nlr.nl:8080/xmlui/handle/10921/927

Van der Sterren, W. (2002). Squad Tactics: Team AI and Emergent Maneuvers. In S. Rabin (Ed.), *AI Game Programming Wisdom* (pp. 233–246). Charles River Media, Inc.