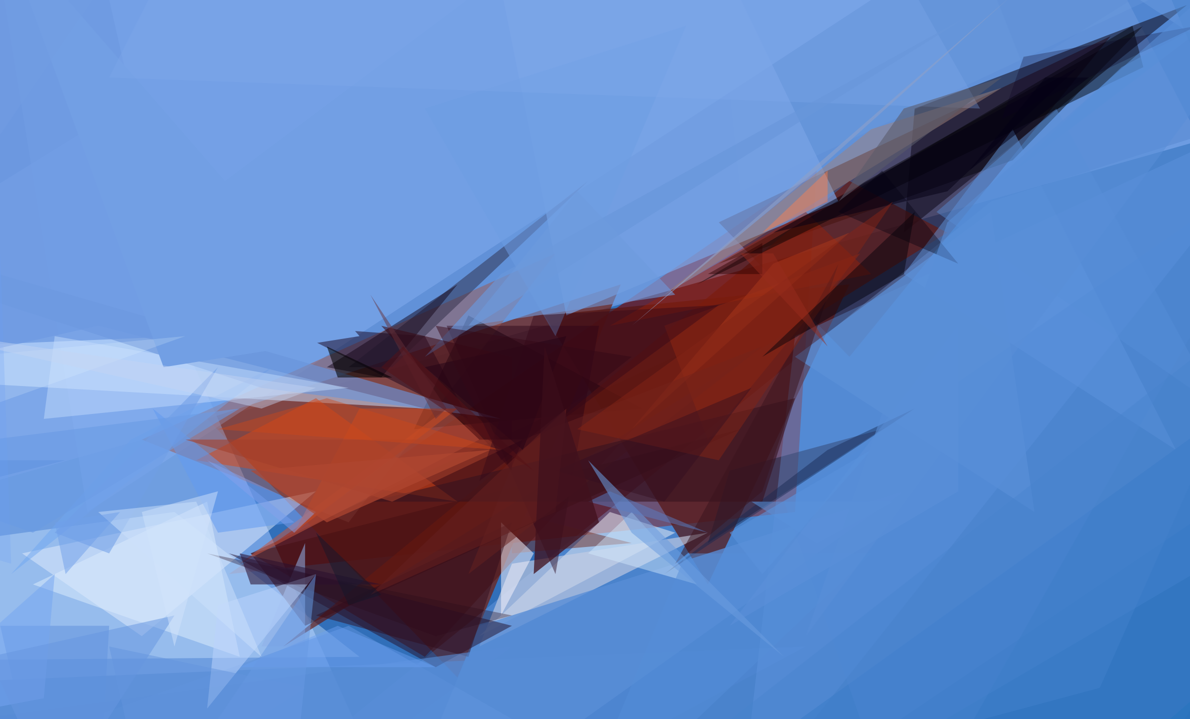
Calculated Moves: Generating Air Combat Behaviour

Toubman, A.

Calculated Moves

Generating Air Combat Behaviour



Armon Toubman

Calculated Moves

Generating Air Combat Behaviour

Calculated Moves

Generating Air Combat Behaviour

PROEFSCHRIFT

ter verkrijging van
de graad van Doctor aan de Universiteit Leiden,
op gezag van Rector Magnificus prof. mr. C.J.J.M. Stolker,
volgens besluit van het College voor Promoties
te verdedigen op woensdag 5 februari 2020
klokke 13.45 uur

door

Armon Toubman
geboren te Amsterdam in 1988

The research reported in this thesis has been funded by and performed at the Royal Netherlands Aerospace Centre NLR, Amsterdam, the Netherlands.

In the first year, the research reported in this thesis has been performed at Tilburg center for Cognition and Communication (TiCC) at Tilburg University, the Netherlands.

The research reported in this thesis has been completed at the Leiden Centre of Data Science (LCDS) hosted by the Leiden Institute of Advanced Computer Science (LIACS) at the Faculty of Science, Leiden University, the Netherlands.

SIKS Dissertation Series No. 2020-01
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Printed by:	Gildeprint, Enschede
Cover/invitation design:	Armon Toubman, derivative of “Dutch F-16 5” by Tony Hisgett. Used under the CC BY 2.0 license. Original photo available at: `https://flic.kr/p/8a6hs4`
Additional credits:	Includes Vista Style People Demo Icons by: `http://www.icons-land.com`

Qapla’

“success!” in Klingon

Preface

Fighter jets are incredible machines. Equally incredible is the skill required to operate them. The pilots of these jets have to excel at observing, communicating, calculating, and making life-or-death decisions while zooming through the sky at inhuman speeds. Excelling at these tasks is only possible by rigorous training. However, defence budget cuts have resulted in fewer aircraft for air forces, and thus fewer aircraft available for real-world training. For instance, the Royal Netherlands Air Force had 213 F-16s available in 1992. Today, in 2019, there are 68 F-16s left. These are planned to be replaced by 37 F-35s in 2023, followed by additional F-35s at a later time. Innovative training methods are thus required to keep fighter pilots ready for future operations.

Another type of incredible machines are modern computers. Over the last decades, computers have become so powerful that they are able to simulate complex virtual worlds, in which humans can interact with life-like virtual entities. The computing power that is available today has also enabled the computers to reinvent their own programming, by means of machine learning algorithms. Since I started my PhD candidacy in 2013, the interest in machine learning has grown exponentially. From credit card fraud detection to self-driving cars, machine learning is now *everywhere*; so much so, that even the latest smartphones have separate processors dedicated solely to machine learning calculations.

One of the most important training tools in the arsenals of air forces is the flight simulator. A simulator relies on virtual entities called *computer generated forces* to create interesting situations that resemble the situations that fighter pilots may encounter in the real world. However, modelling and programming the behaviour of these entities remains challenging. As a result, only few behaviour models are created for the entities, and thus the simulators are left underused. In our research, we put one and one together by applying machine learning to fighter pilot training.

Personally, combining fighter jets and machine learning has felt like turning a piece of science fiction into reality. I hope that the research in this thesis will lead to safer skies. To all fighter pilots training in simulators I extend the greeting used by the Klingon warriors in the *Star Trek* television shows: *Qapla'!*

Armon Toubman
Almere, November 24, 2019

Contents

Nomenclature

AA-REWARD The reward function based on the probability-of-kill of missiles.

BIN-REWARD The binary reward function.

CENT The centralised coordination method with communication.

DECENT The decentralised coordination method with communication.

DOMAIN-REWARD The reward function based on domain knowledge.

$\mathbf{P}_K$ The probability-of-kill of a missile.

TACIT The decentralised coordination method without communication.

List of Acronyms

AI artificial intelligence.

ANOVA analysis of variance.

API application programming interface.

ATACC Assessment Tool for Air Combat CGFs.

BARS behaviourally anchored rating scale.

BOS behaviour observation scale.

BVR beyond-visual-range.

CAP combat air patrol.

CGF computer generated force.

CI confidence interval.

CLI command-line interface.

DIS distributed interactive simulation.

FSM finite-state machine.

GUI graphical user interface.

HSD honest significant difference.

HUD head-up display.

ICC intraclass correlation.

ICP integrated control panel.

IOS instructor operating station.

LCS	learning classifier system.
LWACS	Lightweight Air Combat Simulator.
MEC	mission essential competency.
MFD	multi-functional display.
NLR	Netherlands Aerospace Centre.
PCDS	Personal Computer Debriefing System.
PS	problem statement.
RNLAF	Royal Netherlands Air Force.
RQ	research question.
RWR	radar warning receiver.
SB	Smart Bandits.
TOST	two one-sided t-tests.
WVR	within-visual-range.
XCS	accuracy-based learning classifier system.

List of Definitions

List of Figures

List of Tables

List of Listings

1 Introduction

The military philosopher Sun Tzu once said, “Now the general who wins a battle makes many calculations in his temple ere the battle is fought” (translation by Giles, 1994). Today, we have access to a new type of calculations, which is called *machine learning*. In this thesis, we use machine learning to improve training simulations for air forces.

Air forces are an essential part of modern defence forces. However, air forces worldwide struggle to maintain the combat readiness of their pilots (cf. Ausink, Taylor, Bigelow and Brancato, 2011; Chapman and Colegrove, 2013; Doyle and Portrey, 2014; Church, 2015). In the last decades, shrinking budgets have led to dwindling numbers of operational aircraft, including the aircraft available for training. At the same time, a steady stream of air force deployments has called for pilots at maximum readiness. Maintaining a high level of readiness with smaller numbers of aircraft requires efficient use of alternative means of training, such as simulators (Foster and Fletcher, 2013; Mattingly, Bolton, Walwanis and Priest, 2014; McLean, Lambeth and Mavin, 2016; Bruzzone and Massei, 2017).

Simulators can provide a flexible training environment with access to a wide variety of virtual opponents to train with. Such an opponent is often called a computer generated force (CGF, plural: CGFS), i.e., a computer representation of a real-world force that displays human-like behaviour (cf. Lu and Gong, 2014; Kamrani, Luotsinen and Løvlid, 2016). By training with virtual opponents, trainees can build the experience necessary for air combat operations, at a fraction of the cost of training with real aircraft. In practice however, the variety of virtual opponents is not as wide as it can be. This is largely due to a lack of behaviour models, i.e., computational models used to govern the behaviour that the virtual opponents display (cf. Lu and Gong, 2014; Pelosi and Brown, 2016). The goal of the thesis is to investigate to what extent behaviour models for the virtual opponents in air combat training simulations can be automatically generated, by the use of machine learning.

This chapter is organised as follows. In Section 1.1, we describe the process by which behaviour models are created today. In Section 1.2, we briefly look at the possibility of automatically generating behaviour models. In Section 1.3 we present our problem statement and five research questions. The research methodology is given in Section 1.4. Finally, in Section 1.5 we outline the structure of the thesis.

1.1 The behaviour modelling process

The behaviour modelling process is the process by which behaviour models are created today. For the remainder of this thesis, we define behaviour models as follows.

Definition 1.1 (Behaviour model). A behaviour model is a model that maps (a) the observations made by some entity to (b) the actions that the entity should perform.

We define three roles that take part in the behaviour modelling process: (1) the training specialist, (2) the subject matter expert, and (3) the programmer. We refer to the people that fill in the roles as the professionals[1].

We divide the behaviour modelling process into four steps. The four steps of the behaviour modelling process are shown in Figure 1.1. First, the training specialist writes a behaviour specification for a new CGF. Second, the subject matter expert refines the behaviour specification, using knowledge on the real-world forces that the CGF represents. Third, the programmer creates an executable behaviour model based on the refined behaviour specification. Fourth, the training specialist, subject matter expert, and programmer validate the behaviour model and then make improvements to the model as needed. The four steps are explained in more detail in Chapter 2.

In its current form, the behaviour modelling process brings about two obstacles (see Subsection 1.1.1). Furthermore, we discuss the consequences that the obstacles in the process have for the effectiveness of simulator training (Subsection 1.1.2).

1.1.1 Obstacles in the process

We identify two main obstacles in the behaviour modelling process. We describe them below.

The first obstacle is the observation that the process is *labour-intensive*. However, the professionals capable of performing the four steps in the process come in limited numbers and are not always available. Furthermore, the four steps in the behaviour modelling process must be performed in the order given, and the professionals depend on each other to complete Step 4. Therefore, unavailability of the professionals hampers the completion of the behaviour modelling process.

The second obstacle is the *complexity* of human(-like) behaviour modelling (cf. Banks and Stytz, 2003; Stytz and Banks, 2003a; Stytz and Banks, 2003b). The complexity stems in part from the fact that a CGF's behaviour model has to be able to react as much as possible in a proper way (i.e., as a human would) to all situations that may occur in the simulation (Bourassa, Abdellaoui and Parkinson, 2011). Failure to react appropriately or to react at all to these situations (1) makes the behaviour model brittle, and (2) makes the behaviour produced by the behaviour model to be considered as lacking realism (Bourassa and Massey, 2012). However, eliciting the

[1]See Darken and Blais (2017) for a discussion of the responsibilities of modelling and simulation professionals in the military.

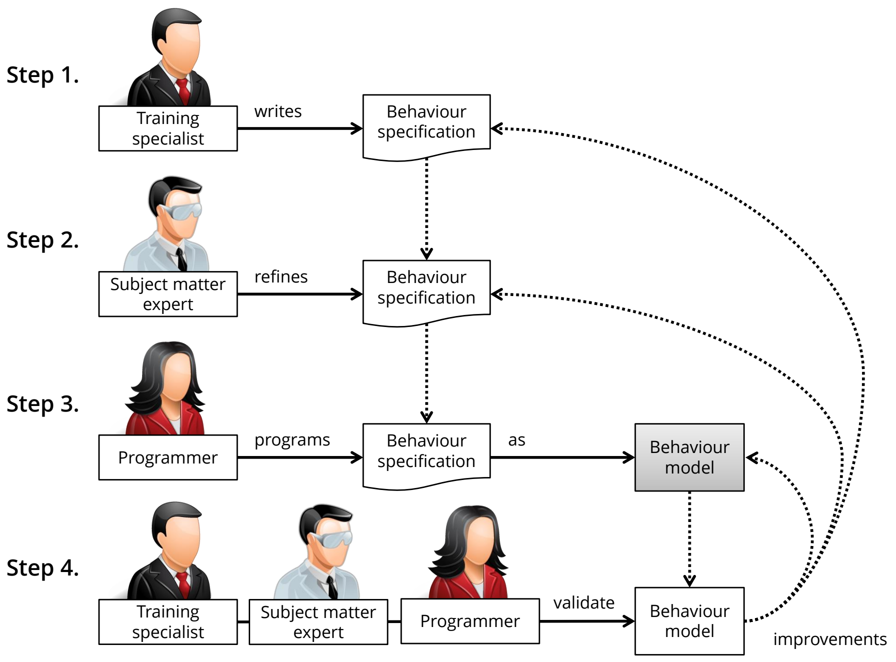


Figure 1.1 The four steps in the behaviour modelling process (adapted from the process described by Gerretsen, Van Oijen, Ferdinandus and Kerbusch, 2017). Step 1: the training specialist writes a behaviour specification for a new CGF. Step 2: the subject matter expert refines the behaviour specification, using knowledge on the real-world forces that the CGF represents. Step 3: the programmer creates an executable behaviour model based on the refined behaviour specification. Step 4: together, the training specialist, subject matter expert, and programmer validate the behaviour model and then make improvements to the model as needed.

knowledge required to model the proper reactions is not a straightforward task (cf. Marcus, 2013; Hoffman, 2014). As a result, behaviour that should have been specified in Step 1 and Step 2 of the behaviour modelling process will only transpire in Step 4. The unspecified behaviour then has to be implemented as an improvement to the model, requiring further work by the professionals.

The two obstacles described above render completing the behaviour modelling process a slow and difficult task. The duration of the process leads to a relatively low number of behaviour models being created. However, at the same time, real-world developments such as (1) new strategies, (2) new tactics, and (3) new equipment are introduced at a high pace. Trainees need to gain experience with these developments in simulations. Therefore, the behaviour modelling process must be reiterated frequently to keep up with the demand for new behaviour models. Furthermore, because of the high pace of real-world developments, the rate of model reuse is low. Lu and Gong (2014) state a reuse rate of behaviour models as low as 10 to 15%.

1.1.2 Consequences for training effectiveness

Because of the low number of behaviour models available for use by CGFs, training specialists are limited in the range of training simulations they can create. A limited range of training simulations has two closely related negative consequences for the effectiveness of the training given by means of these training simulations. We discuss the two consequences below.

The first consequence is that the trainees *miss out on the proven benefits* of variation in training tasks (such as training simulations). For instance, recent studies show that variation in training tasks improves the cognitive and motor skills of trainees (Taylor and Rohrer, 2010; Vakil and Heled, 2016). Furthermore, variation helps trainees to develop the capability to (1) recognise patterns across situations, (2) adapt their mindset to their situation, and (3) come up with creative solutions (Fletcher and Wind, 2014).

The second consequence is that the behaviour of the CGFs becomes *predictable* by the trainees. The predictable behaviour may lead to boredom which transpires in the trainees' behaviour. Furthermore, predictable behaviour may cause the trainees to try to exploit the behaviour of the CGFs, rather than to focus on achieving the learning objectives of the simulations (Lopes and Bidarra, 2011; Silva, do Nascimento Silva and Chaimowicz, 2015).

1.2 Generating air combat behaviour models

The field of artificial intelligence (AI) may offer an alternative to the behaviour modelling process, and improve the effectiveness of training simulations by remedying the two consequences mentioned in the previous section. The alternative is generating behaviour models by means of machine learning. Machine learning programs outperform humans in a variety of tasks (Jordan and Mitchell, 2015), such as credit card fraud detection (Dal Pozzolo, Caelen, Le Borgne, Waterschoot and Bontempi, 2014), cloud computing resource allocation (Hameed, Khoshkbarforoushha, Ranjan, Jayaraman, Kolodziej et al., 2016), and playing games like poker (Bowling, Burch, Johanson and Tammelin, 2015) and Go (Silver, Huang, Maddison, Guez, Sifre et al., 2016; Silver, Schrittwieser, Simonyan, Antonoglou, Huang et al., 2017b). For such tasks, machine learning programs are able to produce creative solutions through a combination of three properties: (1) computational speed, (2) precise constraint satisfaction abilities, and (3) clever learning algorithms. By taking advantage of these three properties and applying the properties to the development of behaviour models, we gain the ability to develop (1) behaviour models at a higher pace, and (2) models with more variation in the behaviour than is currently possible. As a result, the use of machine learning programs to develop behaviour models has the potential to lift the two consequences that the current behaviour modelling process has on training effectiveness.

However, before we apply machine learning to air combat simulations, it is essential to consider the domain of air combat. The domain of air combat is complex, and machine learning methods that operate within this domain must be suited to the challenges posed by the domain. Below, we list five challenges that emerge when generating behaviour models for use in air combat simulations (Subsection 1.2.1). Next, because air combat is a broad concept, we establish the scope of the thesis (Subsection 1.2.2).

1.2.1 Challenges

Below, we identify and describe five challenges: (A) producing teamwork, (B) computationally evaluating CGF behaviour, (C) efficient reuse of acquired knowledge, (D) validating generated behaviour models, and (E) generating accessible behaviour models. The five challenges are not unique to the air combat domain. However, the challenges require solutions that will fit to the domain.

Challenge A: Producing team coordination. Nowadays, the smallest unit that carries out air combat missions consists of two. Flying in pairs has major advantages over flying alone, such as (1) improved situational awareness, and (2) the ability for one teammate to apply offensive pressure on opponents while the other teammate is forced to make defensive manoeuvres (cf. Shaw, 1985; Stillion, 2015). The challenge is to *make optimal use of these advantages* in simulations. It requires a form of coordination between the teammates. Thus challenge A is to let the machine learning method also generate the required team coordination between the CGFs that use the models.

Challenge B: Computationally evaluating CGF behaviour. Machine learning methods require the ability to evaluate the behaviour produced by the behaviour models they generate. In reinforcement learning, which is the family of machine learning methods that we focus on in this thesis (see Chapter 2), the evaluation is performed computationally by the reward function. The reward function is named so because it rewards CGFs for good behaviour, with the intent to stimulate that behaviour (viz. produce better behaviour models). However, the evaluation of air combat behaviour suffers from two issues. First, the concept of *good air combat behaviour* remains ill-defined. Second, non-deterministic factors influence the success of the behaviour of air combat CGFs. We expand on these two issues in Chapter 4. Reward functions that are used in the air combat domain therefore must take into account the two issues in order to stimulate good behaviour with rewards. Thus challenge B is the computational evaluation of the behaviour displayed by air combat CGFs, with the goal of improving the behaviour models generated for these CGFs.

Challenge C: Efficient reuse of acquired knowledge. During the automated generation and testing of behaviour models for a CGF, the machine learning method learns which actions of the CGF are effective in which situations. Therefore, it can be said that the machine learning method acquires and stores knowledge about air combat. It is imaginable that some of this knowledge will be applicable to multiple scenarios in which the CGF may be active. Reuse of air combat knowledge will save computational resources in the search for effective behaviour models for the CGF across different scenarios. Challenge C is *enabling the machine learning method* used to generate behaviour models *to efficiently reuse air combat knowledge that has been acquired previously*.

Challenge D: Validating generated behaviour models. Just like behaviour models that are manually developed by professionals, behaviour models that are generated by a machine learning method have to be validated. Validation of behaviour models ensures that the behaviour models are fit for their intended purpose. Challenge D is *validating behaviour models* that have been generated by means of machine learning, to prove that the models are fit for use in training simulations.

Challenge E: Generating accessible behaviour models. The creative capabilities that machine learning methods possess have a drawback. In brief, the solutions created by these methods may become too clever, and take on forms that are too difficult for humans to comprehend and validate. This is especially problematic for CGF behaviour models, as these models must represent the behaviour of real-world forces at all times. Furthermore, the professionals may wish to inspect and revise behaviour models that have been generated, e.g., in order to slightly adjust the model to better support a learning objective. These professionals are only able to do so if the generated models are constructed in a way that is easily understandable (see, e.g., Luotsinen, Kamrani, Hammar, Jändel and Løvlid, 2016). For this reason, challenge E is *generating accessible behaviour models* by means of machine learning. We define this accessibility as follows.

Definition 1.2 (Accessible behaviour model). A behaviour model is accessible if it is directly interpretable by the professionals (i.e., training instructors, subject matter experts, and programmers) who develop and apply the model.[2]

On various occasions, research has already specifically been focused on the use of machine learning to generate air combat behaviour models. However, as we will show in Chapter 2, the machine learning methods that have been used so far have produced inaccessible behaviour models. In Chapter 2, we will review a machine learning method called dynamic scripting, that was introduced in the previous decade (Spronck, Ponsen, Sprinkhuizen-Kuyper and Postma, 2006). Dynamic scripting was designed to directly address the issue of accessibility as we have described it here. We will investigate dynamic scripting's applicability to air combat simulations.

1.2.2 Scope of the thesis

Both air combat and training simulations are complex domains. It means that in our research we will not take the full domains into account. Below, we restrict the scope of this thesis regarding three areas: (1) the mode of air combat that we study, (2) the specific type of training simulations that we consider, and (3) the width of our view on training simulations.

First, air combat is often divided into two modes (Shaw, 1985). Mode (a) is within-visual-range (WVR) air combat, also known as dog-fighting. In a WVR air combat situation, the opposing aircraft engage each other in the visual arena using on-board cannons and short-range missiles. Mode (b) is beyond-visual-range (BVR) air combat. In BVR air combat, opposing aircraft engage each other using medium-range to long-range missiles, while sensing each other using radar and other instruments. Simulations of the two modes of combat require CGFs with different behaviour. Today, the majority of air combat engagements are BVR engagements (Stillion, 2015; Floyd, Karneeb, Moore and Aha, 2017). For this reason, we restrict the scope of this thesis to BVR engagements, i.e., mode (b).

Second, because our main goal is generating behaviour models for use in air combat training simulations, we specify the particular type of training simulations that we consider in our research. The CGFs that use the generated behaviour models will need to support the learning objectives of this type of training simulations. In this thesis, we restrict ourselves to tactical training at the unit (squadron) level. In tactical training, the objective of the trainees is to defeat all opposing CGFs. The CGFs in tactical training simulations require behaviour models that are capable of handling the most common elements of air combat (e.g., acquiring and pursuing targets, firing missiles, and evading incoming missiles).

Third, training simulations are highly complex systems. The study of training simulations lies at the crossroads of multiple fields of research, e.g., knowledge representation, instructional

[2]Doyle and Portrey (2014) take the definition of accessibility a step further, and pose that the behaviour produced by the models should be "transparent to users not involved in the core modeling process."

theory, human factors, interaction design, competency development, and the modelling of systems, organisations, and behaviours. There are many interactions between these fields. For instance, (a) the development of competencies by the trainees depends on the interaction with the CGFs, (b) the interaction of trainees with CGFs depends on the behaviour models of the CGFs, (c) the behaviour models of the CGFs depend on the modelling technique, knowledge representation, and so on. To study such a chain of interactions as a whole is a complex task that is to be considered intractable with the current means of research and the time allotted to our research project. Therefore, in the thesis, we restrict ourselves to modelling air combat behaviours (see Chapters 3 to 5) and evaluating the perception of the modelled behaviours by training specialists (see Chapters 6 and 7).

1.3 Problem statement and research questions

The two consequences that the current behaviour modelling process has for training effectiveness (Section 1.1), and the prospect of automatic and fast generation of varied behaviour models (Section 1.2) lead us to the following problem statement.

Problem statement: *To what extent can we use dynamic scripting to generate air combat behaviour models for use in training simulations, in such a way that the five challenges of generating air combat behaviour models are met?*

The use of dynamic scripting would bring us underway to meet challenge E. However, challenges A–D remain to be met. Below, five research questions are formulated based on the remaining challenges. In combination, the answers to the five research questions form the answer to the problem statement.

Meeting the first challenge requires investigating the possibility of using dynamic scripting to generate behaviour models that (1) take into account the presence of teammates, and (2) are able to coordinate their observations and actions with these teammates in some manner. This leads us to the first research question.

Research question 1: *To what extent can we generate air combat behaviour models that produce team coordination?*

Dynamic scripting uses a reward function to evaluate the behaviour displayed by the air combat CGFs that use the generated behaviour models. The rewards produced by the reward function are used to adjust newly generated behaviour models in the search for an optimal model. As mentioned (see challenge B), the evaluation of air combat behaviour suffers from two issues. In the literature, these two issues are known as *sparse rewards* and *unstable rewards*, respectively (see Chapter 4). Still, reward functions for air combat behaviour that have been presented in the literature do not always take these two issues into account. However, doing so might lead to behaviour models that produce a more desirable behaviour. This leads us to the second research question.

Research question 2: *To what extent can we improve the reward function for air combat CGFs?*

Dynamic scripting stores the knowledge that a CGF builds throughout the CGF's learning process in the form of weight values that are attached to the rules in the rulebase. The weight value of each rule indicates the rule's importance relative to the other rules in the rulebase. In terms of reuse, it may be possible that the knowledge that is built in one air combat scenario, may also be applied effectively in another air combat scenario. We place the reuse of knowledge in the context of *transfer learning*, i.e., letting a CGF learn in one scenario, and then *transferring* its knowledge to a CGF in a new, unseen scenario. This leads us to the third research question.

Research question 3: *To what extent can knowledge built with dynamic scripting be transferred successfully between CGFs in different scenarios?*

We aim for the generated behaviour models to be used in training simulations. Validating the models is an important step in achieving a productive use of the models. The importance of validation is illustrated by Step 4 in the behaviour modelling process. However, since there is no *one-size-fits-all* solution to the validation of behaviour models, we first have to determine the proper way to do so. This leads us to the fourth research question.

Research question 4: *How should we validate machine-generated air combat behaviour models for use in training simulations?*

The answer to research question 4 is a validation procedure. By means of the procedure, we are able to determine the validity of the behaviour models that we generate in our research. The chosen research approach leads us to the fifth research question.

Research question 5: *To what extent are air combat behaviour models generated by means of dynamic scripting valid for use in training simulations?*

Answering these five research questions will allow us to answer the problem statement. The next section describes the methods that will be used in our research to answer the five research questions.

1.4 Research methodology

In our research, we use four methods to answer the research questions: (1) literature review, (2) automated simulations, (3) questionnaires, and (4) human-in-the-loop simulations. We describe the four methods briefly below.

Literature review. We review scientific articles, books, and technical reports that are related to (1) air combat training simulations (see Section 1.1), (2) the use of machine learning in these simulations (Chapter 2), and (3) the five research questions (Chapters 3–7).

Automated simulations. By automated simulations we mean software simulations in which one team of CGFs engages another team of CGFs in air combat encounters. In this case, both teams of CGFs employ behaviour models for their behaviour. Because such simulations are implemented purely in software, they have the advantages of (a) being able to run faster than real-time and (b) not being dependent on the presence of human participants.

Questionnaires. While reward functions are capable of evaluating the behaviour of a CGF to a certain extent, the final word on the desirability of a CGF's behaviour comes from the training specialist. In the end, it is the training specialist who has to use the CGF in training simulations. Measuring the desirability of a CGF's behaviour is therefore an essential part in the validation of behaviour models. We will develop a novel questionnaire, which we call the Assessment Tool for Air Combat CGFs (ATACC). The ATACC aims to capture the opinions of training specialists observing the behaviour of air combat CGFs, in such a way that we are able to draw conclusions on the desirability of the behaviour.

Human-in-the-loop simulations. By human-in-the-loop simulations we mean simulations in which a team of CGFs using behaviour models engages a team of CGFs controlled by human participants. Training simulations are a prime example of human-in-the-loop simulations. In human-in-the-loop simulations, we are able to observe (1) the behaviour of human pilots, when confronted with CGFs using generated behaviour models, and (2) the behaviour of CGFs using generated behaviour models, when confronted with human pilots.

Below, we briefly describe where we apply the four methods. Table 1.1 gives a summary of the use of the four methods to answer the research questions.

Table 1.1 Research methods used to answer the research questions (RQs).

Method	RQ1	RQ2	RQ3	RQ4	RQ5
Literature review	✓	✓	✓	✓	✓
Automated simulations	✓	✓	✓		✓
Questionnaires					✓
Human-in-the-loop simulations					✓

The literature review is used to answer research questions 1, 2, 3, 4, and 5. Furthermore, we use two simulators for our research: (1) the Lightweight Air Combat Simulator (LWACS) (see Appendix A) and (2) the Netherlands Aerospace Centre NLR's Fighter 4-Ship simulator (see Appendix D). Automated simulations in Lightweight Air Combat Simulator (LWACS) are used to answer research questions 1, 2, and 3. Furthermore, we present the ATACC in Chapter 6. The

ATACC is developed as part of the answer to research question 4, and is then used to answer research question 5. Additionally, both (1) automated simulations and (2) human-in-the-loop simulations in the Fighter 4-Ship are used to answer research question 5.

1.5 Structure of the thesis

This thesis contains eight chapters. Table 1.2 shows which chapters will answer the respective research questions.

Table 1.2 Answering the problem statement (PS) and the research questions (RQs) per chapter.

Chapter	PS	RQ1	RQ2	RQ3	RQ4	RQ5
1	∼	∼	∼	∼	∼	∼
2	∼	∼	∼	∼	∼	∼
3		√				
4			√			
5				√		
6					√	∼
7						√
8	√	√	√	√	√	√

∼ contributes to answer, √ answers

In Chapter 1 we introduce our problem statement and five research questions. Furthermore, the research methodology by which the research questions are addressed is presented.

In Chapter 2, we provide background information from the literature (see also Section 1.1) on four topics: (1) details of the steps in the behaviour modelling process, (2) the potential benefits and drawbacks of the use of machine learning in training simulations, (3) past approaches to using machine learning for generating air combat behaviour models, and (4) dynamic scripting and its applicability to air combat simulations.

In Chapter 3, we introduce three methods for team coordination: (1) TACIT, (2) CENT, and (3) DECENT. We investigate how beneficial the team coordination methods are by means of an experiment, and then answer research question 1.

In Chapter 4, we zoom in on a specific part of the dynamic scripting process, viz. the reward function. We show how the use of three distinct reward functions influences the behaviour of our CGFs, and then answer research question 2.

In Chapter 5, we investigate to what extent the knowledge that is built by a CGF in some air combat scenario can be transferred successfully to a CGF in a different air combat scenario, and then answer research question 3.

In Chapter 6, we design a validation procedure by which behaviour models that are generated for air combat CGFs may be validated. Furthermore, we present the ATACC, and then answer research question 4.

In Chapter 7, we apply our validation procedure to newly generated behaviour models in the Fighter 4-Ship simulator, and then answer research question 5.

In Chapter 8, we conclude the thesis by providing a summary of the answers to the five research questions. Finally, based on these answers, we formulate the answer to the problem statement. Thereafter we present two recommendations for future work.

2 Foundations

In this chapter, we will discuss four topics that are related to our research. First, we will have a detailed look at the steps of the behaviour modelling process (Section 2.1). Second, we will discuss the potential benefits and drawbacks of the use of machine learning on training simulations (Section 2.2). Third, we will describe the three categories of machine learning tasks (Section 2.3). Furthermore, we will take a closer look at reinforcement learning, and the dynamic scripting technique. Fourth, we will give an overview of past research on generating air combat behaviour with machine learning (Section 2.4). Finally, we will conclude the chapter by a summary (Section 2.5).

2.1 The steps in the behaviour modelling process

In this section, we describe the four steps of the behaviour modelling process in detail. The four steps were briefly mentioned in Chapter 1 (see Figure 1.1). The description of the steps below serves to create a better understanding of (1) how modern behaviour models are created, (2) the dependencies in the process that lead to the obstacles (see Subsection 1.1.1), and (3) the context in which a machine learning solution for behaviour generation would operate.

Step 1. The *training specialist* identifies training needs, resulting in a collection of learning objectives (cf. Stacy and Freeman, 2016). To help trainees reach these learning objectives in the simulator, the training specialist requires a CGF with certain behaviour to interact with the trainees. The training specialist writes a behaviour specification containing the required behaviour. As a case in point, consider a training specialist who has set a learning objective for successfully evading long-range missiles. For this learning objective, the training specialist requires a CGF that fires long-range missiles, and therefore writes "long-range missile firing behaviour" in the behaviour specification.

Step 2. The *subject matter expert* refines the behaviour specification that was written by the training specialist. The subject matter expert knows the behaviour of the real world forces represented by the CGF, such as (1) the technical capabilities of the CGF's aircraft, and (2) the principal strategies and tactics of the represented real world forces (see, e.g., Løvlid, Alstad, Mevassvik, De Reus, Henderson et al., 2013). Returning to the example from Step 1, the subject matter expert refines the behaviour specification by, e.g., (1) defining the specific type of missile that the CGF should fire, and (2) adding any manoeuvres that the CGF should make before and after firing.

Step 3. The *programmer* takes the refined behaviour specification and implements the specification as a computer model. We call the resulting program the behaviour model. The behaviour model commonly takes one of three executable forms: (1) the form of a script (cf. Abdellaoui, Taylor and Parkinson, 2009; Toubman, Roessingh, Van Oijen, Hou, Luotsinen et al., 2016a), (2) a finite-state machine (cf. Fu, Houlette and Jensen, 2003; Coman and Muñoz-Avila, 2013), or (3) a behaviour tree (cf Khatami, Huibers and Roessingh, 2013; Marzinotto, Colledanchise, Smith and Ögren, 2014; Zhang, Sun, Jiao and Yin, 2017). When loaded in a simulator, the behaviour model governs the behaviour of the CGF. In other words, the behaviour model selects and implements the CGF's actions, based on the observations that the CGF makes in the simulator. However, simulators usually only provide CGFs with a limited set of possible observations and actions. It is the programmer's responsibility to interpret the behaviour specification, and then to accurately translate the specified behaviour into an executable form using the possible observations and actions as provided by the simulator. For example, consider (1) a behaviour specification that calls for a CGF that patrols a section of the airspace, and (2) a simulator that only provides actions by which a CGF can set its own altitude and heading. In this example, the programmer has to interpret the specified patrol and translate it to the correct sequence of altitude and heading settings.

Step 4. The *professionals* work together to validate the behaviour model. Multiple methods exist for validating behaviour models, each method with its own advantages and disadvantages (cf. Petty, 2010; Birta and Arbez, 2013). Validating the behaviour models commonly involves testing and reviewing the behaviour produced by the behaviour models, when used by CGFs. Each of the three professionals validates the behaviour model from their own viewpoint: (1) the training specialist determines whether the behaviour model can be adequately used to help trainees reach their learning objectives, (2) the subject matter expert decides whether the displayed behaviour matches the behaviour of real-life forces, and (3) the programmer establishes whether his[1] interpretation of the behaviour specification was correct with help from the training specialist and the subject matter expert. Finally, the professionals work out and implement improvements to the behaviour model. After any improvements have been implemented, the behaviour model is ready to be used by a CGF in a training simulation.

[1]For brevity, we use "he" and "his" whenever "he or she" and "his or her" are meant.

2.2 Machine learning in training simulations: potential benefits and drawbacks

Because machine learning is a powerful technology, it is important to carefully consider the impact it has on each application domain. In this section, we look at the possible impact of the use of machine learning on training simulations in terms of the potential benefits (Subsection 2.2.1) and the potential drawbacks (Subsection 2.2.2). Note that we purposefully do so before we describe machine learning itself (see Section 2.3). It enables us to confine the description to details necessary for a good understanding of our research.

2.2.1 Potential benefits

Below, we identify three potential benefits that appear when using machine learning to generate air combat behaviour models. They are: (1) faster development of behaviour models, (2) detection of patterns, and (3) online behaviour adaptation.

The first potential benefit is *faster development of behaviour models* (cf. Doyle, Watz and Portrey, 2015; Oswalt and Cooley, 2019). Computers excel at (1) storing and retrieving knowledge, and (2) calculation. Therefore, given (1) a database that stores the knowledge of a subject matter expert, and (2) a suitable machine learning algorithm, a computer may be able to automatically generate a correct behaviour model based on a behaviour specification (Stytz and Banks, 2003b). Automating the behaviour development process makes the process less dependent on the availability of two of the required professionals, i.e., the subject matter expert and the programmer. Furthermore, the ability to develop behaviour models at high speed enables the training specialist to support learning objectives with multiple and varied behaviour models.

The second potential benefit is the *detection of patterns* in behaviour. Faster development of behaviour models enables training specialists to see how trainees react to CGFs that behave in varying manners. Large data sets of these reactions allow for computer programs by which training specialists can detect patterns in the behaviour of trainees (e.g., areas of improvement for the trainees), and then adjust the current learning objectives to the needs of the trainees (cf. Mittal, Doyle and Watz, 2013; Sottilare, 2013; Ososky, Sottilare, Brawner, Long and Graesser, 2015; Oswalt and Cooley, 2019). Using data sets to improve training as described above is known as educational data mining, adaptive tutoring, and adaptive training (cf. Peña-Ayala, 2014; Goldberg, Davis, Riley and Boyce, 2017). Furthermore, large data sets of behaviour in simulations allow searching for exploitable patterns in the tactics that are taught to trainees, by the use of machine learning and big data techniques. This practice is known as computational red teaming (cf. Abbass, Bender, Gaidow and Whitbread, 2011; Wang, Shafi, Ng, Lokan and Abbass, 2017).

The third potential benefit is *online behaviour adaptation*. With faster development of behaviour models, it becomes possible for a computer to change the behaviour of CGFs while a training simulation is taking place (viz. online) (cf. Olde and DiCola, 2014; Oswalt and Cooley, 2019).

By adapting the behaviour of a CGF to the behaviour of the trainee in the training simulation, the CGF is able to continuously challenge the trainee (Lopes and Bidarra, 2011). A similar use of machine learning is being investigated in the field of video games (i.e., tuning the behaviour of non-player characters to the player, while the player is playing the video game) (cf. Yannakakis and Togelius, 2014), a field that is strongly tied to the field of military simulations (Smith, 2010).

In our research we will focus on materialising the first potential benefit. It will give a sufficient insight into the impact of machine learning on training simulations.

2.2.2 Potential drawbacks

Already in 2003, Petty (2003) identified the potential drawbacks of using automatically generated behaviour models in simulations for training, analysis, and experimentation purposes. Below, we summarise Petty's work into what we consider to be the two main drawbacks for training simulations, namely (1) the emergence of unrealistic behaviour, and (2) the resulting loss of training control.

Petty (2003) warns against the use of machine learning programs with the goal of automatically generating behaviour models for use in training simulations, as the models may produce unrealistic behaviour. The behaviour of a CGF that represents a particular force (e.g., a specific military branch of a specific nation) should accurately follow the doctrine of that force, in order to be perceived as realistic (cf. Doyle and Portrey, 2014; Bolton, Tucker, Priest, McLean, Beaubien et al., 2016). The doctrine of a military force is a "guide to action", viz. a handbook for conducting operations that describes how and when the different capabilities of the military should be put to use (see, e.g., Paparone, 2017). Petty uses the term *doctrinal behaviour* to refer to the behaviour of CGFs that appear to follow their doctrine.

If a CGF fails to follow its doctrine, but the CGF still acts as though it could be operated by a human, then the CGF exhibits what Petty (2003) calls *non-doctrinal behaviour*. Such behaviour may be the result of a machine learning program that is trying to optimise behaviour models regarding some constraints. Although non-doctrinal behaviour may be physically realistic, real-world forces are unlikely to display such behaviour. In extreme cases, the behaviour of the CGF may surpass the capabilities of a human operator and become *non-human behaviour*. An example of non-human behaviour is the display of inhumanly fast reaction times to threats.

The emergence of unrealistic behaviour (in the form of either non-doctrinal behaviour, or non-human behaviour) may lead to a loss of training control. As a machine learning program changes the behaviour of CGFs away from doctrinal behaviour, it becomes possible that the CGFs no longer support the training goals that the training specialist has set for a particular simulation. Thus, new tools are required to keep the creative power of machine learning techniques under a permanent check (cf. Shaffer, Ruis and Graesser, 2015; Wray, Woods, Haley and Folsom-Kovarik, 2017). In our research, we aim to mitigate the potential loss of training control by the development of a proper validation method for generated behaviour models (see Chapters 6 and 7).

2.3 Machine learning

So far, we have mentioned machine learning without explaining its details. In this section, we introduce the three categories of machine learning tasks (Subsection 2.3.1). Next, we take a detailed look at reinforcement learning, one of the categories (Subsection 2.3.2). Finally, we discuss dynamic scripting, a reinforcement technique (Subsection 2.3.3).

2.3.1 The three categories of machine learning tasks

Machine learning tasks are commonly split up into three categories: (1) supervised learning, (2) unsupervised learning, and (3) reinforcement learning (cf. Bishop, 2006; Alpaydin, 2010; Jordan and Mitchell, 2015). We briefly describe the three categories of machine learning tasks below. Here, we separate the tasks from the specific techniques that are used to perform the tasks. For instance, neural networks and deep learning techniques can be used to perform tasks in each of the three categories.

Category 1: Supervised learning tasks. Supervised learning tasks are tasks in which the computer has to learn a function that maps the inputs to the desired outputs. To learn this function, the computer receives a data set containing the function's inputs and desired outputs. The *classification* of data (for instance, credit card fraud detection) is an example of a supervised learning task (see, e.g., Dal Pozzolo et al., 2014). The computer is provided with a data set containing credit card transactions that are labelled by human experts as *fraudulent* or *regular*, and then learns a function to identify fraudulent transactions. When the computer is given new transactions, it can detect fraudulent transactions using the function it has learnt.

Category 2: Unsupervised learning tasks. Unsupervised learning tasks are tasks in which the computer has to create automatically a model of the data it receives. In unsupervised learning tasks, there is no desired output. Instead, it is the newly created model that is the output of interest. The model captures the structure of the data in ways that human experts may not have foreseen. An example of an unsupervised learning task is *clustering*. Clustering is the grouping of data points. The computer learns a model of the most important properties of the data points, and divides the data points into a number of groups according to these properties. The groups capture a hidden structure in the data that was given as input. Unsupervised learning tasks such as clustering are commonly part of exploratory data analysis (see, e.g. Peña-Ayala, 2014).

Category 3: Reinforcement learning tasks. Reinforcement learning tasks are tasks in which an agent (e.g., a program or robot) that has to learn to act in some environment. The environment provides *rewards* to the agent when the actions of the agent affect the environment in some desirable way. In most reinforcement learning tasks, the environment is dynamic. For example, each action performed by the agent changes the environment in some way. As a result, (1) the future states of the environment, (2) the actions that are possible in these future states, and (3) the rewards that are obtained because of these possible actions all depend (at least partially) on the agent's current actions. Therefore, the computer can only learn which actions lead to the most rewards by actually interacting with its environment. The desired result of reinforcement learning is a sequence of actions that leads to the most rewards.

An example reinforcement learning task is learning to play the video game *StarCraft II* (Blizzard Entertainment, 2010). The goal of playing this game is to defeat the (virtual or human) opponent. The actions of the agent are, for instance, (a) creating buildings and troops, and (b) using troops to attack the buildings and troops of the opponent. It is up to the agent to learn what to build and how to properly instruct the troops to make their attacks (see, e.g., Lee, Tang, Zhang, Xu, Darrell et al., 2018).

Given the three categories of machine learning tasks, how should the task of generating air combat behaviour be approached? An important consideration is the availability of data to learn from. Real-world air combat data sets are difficult to obtain because of their military nature. Furthermore, if such a data set were available, it is likely that it would not contain sufficient examples for a machine to learn a generalised model of air combat behaviour from. Air combat situations are subject to what is known as the *curse of dimensionality* (originally introduced by Bellman, 1957; see also Roessingh, Rijken, Merk, Meiland, Huibers et al., 2011; Liu and Ma, 2017). Air combat has so many variables (e.g., the number of participating aircraft, their positions, headings, speeds) that it is unfeasible to enumerate and label all possible states. Additionally, the labelling of air combat behaviour as desirable or not remains an open problem (see Chapter 7).

Simulation technology enables us to create an air combat data set on demand. By treating the task of generating air combat behaviour models as a reinforcement learning task within a simulation, the reinforcement learning agent is able to gradually explore the state space in the search for desirable behaviour. We supply the agent with a simulated fighter jet and a restricted set of actions that the agent can perform at any moment (e.g., changing heading or firing a missile). This way, the agent's creativity is not bound to a limited set of real-world examples of behaviour, but still restricted to a particular set of actions that it can perform. In the next subsection, we discuss reinforcement learning in detail.

2.3.2 Reinforcement learning

Reinforcement learning is "learning what to do – how to map situations to actions – so as to maximize a numerical reward signal" (Sutton and Barto, 1998, chap. 1). Below, we briefly review the most important concepts in reinforcement learning. We base our review on the works by Sutton and Barto (1998), Heidrich-Meisner, Lauer, Igel and Riedmiller (2007), Alpaydin (2010), Grondman, Busoniu, Lopes and Babuska (2012) and Arulkumaran, Deisenroth, Brundage and Bharath (2017).

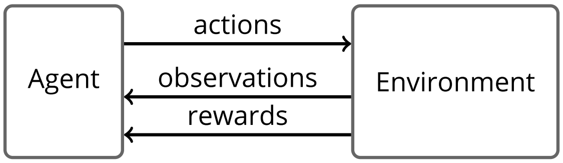


Figure 2.1 The reinforcement learning loop.

Descriptions of reinforcement learning commonly start by showing the reinforcement learning loop (see Figure 2.1). This loop shows the interaction between the agent and its environment. The agent performs an action. This action has some effect on the state of the environment. The environment provides a reward to the agent. Next, the agent observes the new state of the environment, and selects a new action to perform.

The agent selects its actions by means of its *policy* (i.e., its action plan). The policy is equivalent to what we so far have called the behaviour model. The agent learns by changing its policy in such a way that it can expect to receive more rewards in the future. The changes to the policy are guided by the rewards collected by the agent. However, the agent is never sure which specific action has lead to the rewards that the agent receives, as the reward may be the result of an earlier action. This is known as the *credit assignment problem*. The credit assignment problem is especially present in tasks where the reward is only presented to the agent after a sequence of actions (i.e., *delayed rewards*). Alpaydin (2010) summarises the use of rewards by noting that the agent is not dealing with a *teacher* that shows the agent how to act (such as in supervised learning), but rather with a *critic* that tells the agent how well it is doing. However, as Alpaydin (2010, chap. 18) notes, “the feedback from the critic is scarce and when it comes, it comes late.”

The goal of collecting rewards presents a dilemma to the learning agent. Consider a game with two possible actions, action a and action b. Earlier, the agent performed action a, and then received a reward of $+50$. Therefore, action a is said to have a *value* of $+50$ to the agent. If the agent has not yet tried action b, that action will have a value of 0. The *state-action value* is the estimated reward the agent can expect to receive by performing that action in that state. The function that estimates the state-action values is called the *value function*. The agent’s dilemma is as follows: Should the agent *exploit* action a to continue receiving rewards, or should it *explore* the use of action b, to see if action b leads to a higher reward than action a? Of course, by performing action b the agent also risks receiving (1) a lower reward or (2) no reward at all. Exploitation maximises the expected rewards in the short term, but exploration may yield more rewards in the long term (Sutton and Barto, 1998). Reinforcement learning methods that only allow the learning agent to exploit known action-values are called *greedy* methods. Alternatively, the agent can be allowed to explore some of the time. With small probability ϵ, the agent selects an action at random, rather than selecting the action with the highest known state-action value.

Methods that allow exploration in this manner are called *ϵ-greedy* methods.

Moreover, in reinforcement learning an important distinction is made between three classes of techniques: (1) actor-only techniques, (2) critic-only techniques, and (3) actor-critic techniques. Each class of techniques uses value functions and policies in different ways. We describe the three classes of techniques below.

First, *actor-only* (also called *policy-only*) techniques learn without a value function. Instead, they employ parametrised policies, and use any obtained rewards to follow a gradient descent over the parameters towards more rewards. Because of the parametrised policies, actor-only techniques are capable of learning policies in continuous action spaces. However, because of the use of gradient descent, information from earlier interactions with the environment is not retained (i.e., there is no real *learning* taking place).

Second, *critic-only* (also called *value-only*) techniques work by estimating the values of all states. Afterwards, the value function is used to derive the policy that is estimated to lead to the most reward. Critic-only methods learn optimal value functions, but the resulting policies are not guaranteed to be optimal (Grondman et al., 2012).

Third, *actor-critic* techniques try to combine the best features of critic-only and actor-only methods. In actor-critic techniques, the actor and the critic are modelled separately. Initially, the actor performs its actions based on some random policy. The critic detects the rewards that are received, and updates its value function. Next, the actor changes its policy based on the updated value function. The estimates of the critic lower the variance in the actor's policy updates. As a result, the combination of a separate actor and critic has two advantages: (1) increased learning speed and (2) good convergence properties compared to actor-only and critic-only techniques (Grondman et al., 2012). However, the trade-off is that the critic's estimates are inaccurate at the beginning of the learning process, when the critic has not yet seen many states and their values. Still, the two advantages have made actor-critic techniques popular in many domains.

In the last decade, an actor-critic reinforcement learning technique called *dynamic scripting* was introduced (Spronck et al., 2006). Two properties set dynamic scripting apart from other actor-critic techniques: (1) the behaviour models that it generates are accessible, and (2) by letting domain experts specify all relevant state-action pairs, the dynamic scripting is capable of counteracting the curse of dimensionality. In the next subsection, we further discuss the dynamic scripting technique.

2.3.3 Dynamic scripting

Dynamic scripting is a reinforcement learning technique (Spronck et al., 2006). Originally, dynamic scripting was designed as a behaviour generation technique for non-player characters in video games. The design of dynamic scripting was motivated by a discontent with the machine learning techniques that were available at the time (e.g., neural networks, evolutionary algorithms, and Q-learning). Specifically, the available machine learning techniques failed to fulfil eight requirements that, according to Spronck et al. (2006, p. 219), were essential for maintaining the entertainment quality of video games. The eight requirements are as follows.

Requirement 1: Speed. Techniques must be computationally *fast*.

Requirement 2: Effectiveness. Techniques must be *effective* and produce adequate behaviour at all times.

Requirement 3: Robustness. Techniques must be *robust* against the inherent randomness in video games.

Requirement 4: Efficiency. Techniques must be *efficient* in learning, since each encounter between human players and non-player characters will most likely be different. Furthermore, the encounters are sparse, meaning there are few learning opportunities.

Requirement 5: Clarity. The policies that are generated must be easily *interpretable* by experts such as game developers.

Requirement 6: Variety. Techniques must be able to produce *variety* in the generated behaviour, to keep the video games entertaining.

Requirement 7: Consistency. Techniques must produce adequate behaviour from a limited number of learning opportunities with high *consistency*, independent of the behaviour of the player.

Requirement 8: Scalability. The technique must be able to *scale* the difficulty level of the generated behaviour to the skill level of the human player.

Of course, training simulations do not serve to provide entertainment. Still, there are three parallels between training simulations and video games that make the eight requirements relevant to training simulations as well. In both simulations and video games, a human participant (1) controls some avatar of themselves in a (somewhat realistic) representation of the world, and (2) encounters virtual agents that challenge the participant in some way. Furthermore, (3) the participant aims to reach some measurable goal: either setting high scores and completing levels (in video games), or honing and demonstrating his skills (in simulations). Therefore, it is important to investigate dynamic scripting as a technique for generating behaviour models for air combat CGFs.

Spronck et al. (2006) note that the key to fulfilling the eight requirements is the inclusion of domain knowledge in the machine learning technique. For this reason, they made predefined domain knowledge an integral part of dynamic scripting. Below, we review the workings of dynamic scripting, including the use of domain knowledge in the learning process.

The principal unit of behaviour in dynamic scripting is the *behaviour rule*. The policies generated by dynamic scripting are groups of behaviour rules. These groups are called *scripts*. We define behaviour rules and scripts below.

Listing 2.1 Example behaviour rule.

```
observe(radar(opponent)) → act(turn(180));
```

Definition 2.1 (Behaviour rule)**.** A behaviour rule is an *if-then* statement with an observation as the condition of the statement, and an action as the consequence of the statement.

Definition 2.2 (Script)**.** A script is a set of behaviour rules that is used as a policy.

A behaviour rule (henceforth: *rule*) directs an agent (e.g., a CGF) to behave in the following manner: *if* the agent makes the observation stated in the condition, *then* the agent takes the action stated in the consequence. Listing 2.1 shows an example behaviour rule. In Listing 2.1 and the remainder of the thesis, we use the two conventions for writing rules: (1) the condition and the consequence of a rule are separated by the arrow symbol →, and (2) rules end with a semicolon.

In normal words, the rule shown in Listing 2.1 means "if I observe the presence of an opponent using my radar, I turn around 180 degrees." Such a rule only constitutes a limited part of the behaviour that may be desired from a CGF inside simulations. Of course, more rules are required to provide behaviour that is applicable to other air combat situations. A script that only contains the rule shown in Listing 2.1 will cause a CGF to react only to opponents on its radar. However, other behaviour (e.g., offensive behaviour) may also be desired from the CGF.

The rules that dynamic scripting uses to form scripts are stored in a database called the rulebase. We define the rulebase below.

Definition 2.3 (Rulebase)**.** A rulebase is a database with (1) rules and (2) weight values associated to the rules.

The *weight value* (henceforth: *weight*) of each rule is akin to the state-action values that were mentioned previously (see Subsection 2.3.2). The weights of the rules in the rule base indicate the contribution of each rule towards desirable behaviour. Based on the rewards from the environment, dynamic scripting updates the weights of the rules that were used to obtain the rewards. Furthermore, the weight of each rule influences the probability that a rule is selected, whenever dynamic scripting generates a new script. This way, dynamic scripting learns which rules provide the behaviour that leads to the most rewards.

Additionally, each rule has an associated priority value. When two or more rules fire at the same time (e.g., because they have equal or logically overlapping conditions), the priority value is used to determine which rule takes precedence over the other rules. This way, it can be ensured that only the actions from one rule are executed.

The dynamic scripting learning process consists of three steps: (1) rule selection, (2) control,

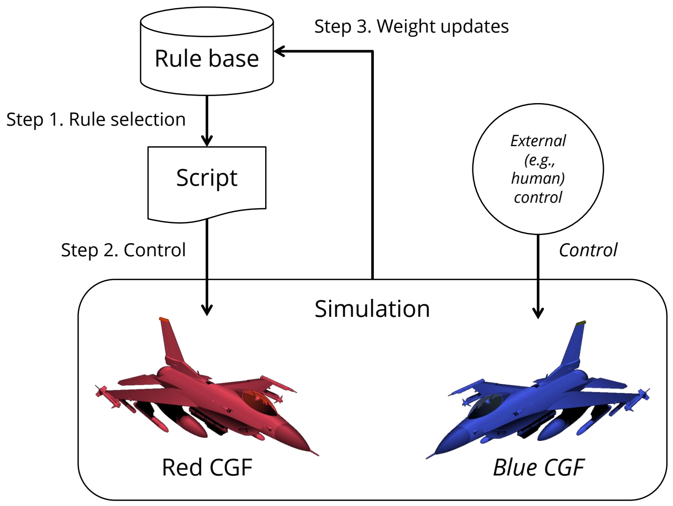


Figure 2.2 The three steps of the dynamic scripting learning process. Adapted from (Spronck, Ponsen, Sprinkhuizen-Kuyper and Postma, 2006).

and (3) weight updates, as shown in Figure 2.2. Below, we describe the three steps of the learning process.

First, dynamic scripting selects n rules from the rulebase. The selected rules form a script. The number of rules per script n depends on the domain (e.g., the number of possible states and actions, and how the states and actions are used in the rules). Therefore, n must be carefully chosen by a domain expert. The rules are selected from the rulebase based on their weights, by means of repeated *roulette wheel selection*. In roulette wheel selection, the probability of selecting a rule is equal to that rule's weight, divided by the sum of all weights in the rulebase.

Second, the script is used to control the behaviour of some agent in its environment. Here, we consider the case of a CGF that inhabits an air combat simulation. The CGF continuously observes its environment using its sensors (see Appendix A). The script checks whether the observations of the CGF match the conditions of one or more rules. Whenever the condition of a rule matches the observations of the CGF, the rule is said to *fire*, and the CGF performs the action that is defined by the rule.

Third, the behaviour of the CGF leads to updates of the weights in the rulebase. The weights are updated by means of two functions: (1) a fitness function, and (2) a weight adjustment function. The *fitness function* evaluates the behaviour that the CGF has displayed, and awards a *fitness value* (viz. a reward) to the CGF. In other words, the fitness function defines what behaviour is desirable (see Chapter 4). For example, the CGF might receive a fitness value of $+1$ if it completes some task, and a fitness value of -1 if it does not. The *weight adjustment function* takes the fitness value of the CGF, and then uses it to calculate the necessary adjustments to the weights in the rulebase. Two mechanisms regulate the weight updates. These mechanisms are (1) restricting the growth of the weights in the rulebase, and (2) keeping the total sum of the weights in the rulebase constant. The growth of the weights is restricted by keeping each weight in the range $[W_{min}, W_{max}]$, where W_{min} and W_{max} are the minimum and maximum weights, respectively. Additionally, when the weights of certain rules must increase, they do so at the cost of the weights of the other rules in the rulebase (and vice versa). The constant *redistribution of weights* (1) allows the weights to converge to a set of well-performing (i.e., high-weight) rules, yet also (2) enables dynamic scripting to rapidly adapt to new situations. When a well-performing rule suddenly ceases to perform well, the weight that is taken from it is redistributed to other rules, thereby immediately increasing the probability that those other rules are selected for a new script.

An important feature that makes dynamic scripting stand out from other reinforcement learning techniques is the use of rules, and in particular, the origin of the rules. Spronck et al. intended for the rules in the rulebase to be manually written, based on domain knowledge. This way, the domain expert can define rules that make sense regarding the domain knowledge, and then let the dynamic scripting algorithm discover the combinations of rules (i.e., the scripts) that lead to the most desirable behaviour. On the nature of the rules, Spronck et al. (2006, p. 221) note that “it is imperative that the majority of the rules in the rulebase define effective, or at least sensible, agent behaviour.”

The use of manually written rules can be viewed as both a drawback and an advantage. On one hand, writing the rules requires costly domain knowledge and human labour. On the other hand, rules only have to be written one time for each class of agent (e.g., CGFs that model a particular combination of pilot and fighter jet). Once the rules are stored in the rulebase, the rules can be used by each agent of that class. Furthermore, the rules are not edited by the dynamic scripting algorithm, and therefore remain accessible to the human professionals. Since the introduction of dynamic scripting, various methods have been introduced that automatically write behaviour rules (see, e.g., Thawonmas and Osaka, 2006; Ponsen, Spronck, Muñoz-Avila and Aha, 2007; Kanetsuki, Thawonmas and Nakata, 2015). While these methods have shown the capacity to generate effective rules (i.e., rules that lead to desirable behaviour), they also threaten the control that the professionals need to have over the resulting behaviour (see Subsection 2.2.2).

The combination of (1) rules and (2) domain knowledge makes dynamic scripting a versatile machine learning method. This versatility is shown by the diversity in the applications that can be found in the literature (see Table 2.1). Dynamic scripting has been used in multiple video game genres, each with distinctive features (e.g., real-time/turn-based, continuous moves/discrete moves, control of one or more agents). The demonstrated versatility of dynamic scripting provides a solid foundation for application in the air combat domain.

Table 2.1 A selection of dynamic scripting applications from the literature.

Application	Sources
Business simulation games	Bijlsma (2014)
Fighting games	Thawonmas and Osaka (2006), Kanetsuki, Thawonmas and Nakata (2015) and Majchrzak, Quadflieg and Rudolph (2015)
First-person shooter games	Policarpo, Urbano and Loureiro (2010)
Platform games	Ortega, Shaker, Togelius and Yannakakis (2013)
Real-time strategy games	Ponsen, Muñoz-Avila, Spronck and Aha (2005) and Dahlbom and Niklasson (2006)
Role-playing games	Spronck, Ponsen, Sprinkhuizen-Kuyper and Postma (2006), Timuri, Spronck and Van den Herik (2007) and Ludwig and Farley (2008)
Turn-based strategy games	Santoso and Supriana (2014)

2.4 Past approaches to generating air combat behaviour

The high stakes involved in air operations have invited multiple generations of computer scientists to support the training of fighter pilots by means of innovative machine learning programs. Furthermore, the complexity of the air combat domain (including the behaviour required of air combat CGFs), makes it an interesting application domain for machine learning algorithms. So far, past approaches to the generation of behaviour models for air combat CGFs has focused on neural networks (Subsection 2.4.1) and evolutionary algorithms (Subsection 2.4.2). Despite the continued interest in air combat behaviour modelling, we are unaware of any standardised tests or benchmarks for the performance of behaviour models for CGFs. The nearest example of such a test is a recent competition (Defense Advanced Research Projects Agency (DARPA), 2019) aimed at the creation of behaviour models for WVR air combat. Therefore, it remains difficult to assess the impact of each individual study performed in the air combat domain.

2.4.1 Neural networks

Neural networks have been applied in various forms to the generation of air combat behaviour. The strength of neural networks is the ability to emulate complex functions by learning from examples. Four of their weaknesses are (1) the need for long training phases, (2) the tendency to strongly converge toward a single solution, (3) trained neural networks are difficult to understand

and reason about, and (4) trained networks are practically impossible to manually edit. In other words, neural networks are powerful yet inaccessible. On multiple occasions, researchers have explored the potential of neural networks in the air combat domain. Below, we discuss four works that apply neural networks to air combat behaviour, viz. the works by (a) Rodin and Amin (1992), (b) McMahon (1990), (c) Teng, Tan and Teow (2013), and (d) Liu and Ma (2017).

Early work with neural networks includes the use of a three-layer back-propagation network by Rodin and Amin (1992) for predicting and countering within-visual-range tactical manoeuvres. With a single hidden layer, Rodin and Amin's network could not "satisfactorily distinguish" a set of simple one-versus-one manoeuvres from two-versus-one manoeuvres. Extensive testing of different architectures of the network resulted in a network with two hidden layers. This research exposes the third weakness of neural networks, which is the difficulty of reasoning about its construction. So far, trial and error has been the best way of finding optimal networks. Furthermore, Rodin and Amin report "successfully training" their network after 60,000 iterations.

Second, McMahon (1990) trained a neural network to recognise within-visual-range situations and choose appropriate manoeuvres. The neural network learned from examples. After 17,500 iterations, the network had learned to classify 36 out of 38 situations correctly. The network's classification capability was compared to that of a rule-based system containing expert knowledge. McMahon found that the neural network was able to classify situations correctly 2.5 times more often than the rule-based system. The high rate of correct classification was attributed to the generalising capability of neural networks. The capability to generalise enables neural networks to classify situations with noisy or incomplete data. Such situations are hard to classify for rule-based systems, unless the ability to deal with noisy or incomplete data is explicitly coded from the knowledge that is elicited from experts. Recently, research into new methods for the classification of air combat situations has been continued by Alford, Borck, Karneeb and Aha (2015).

Third, Teng et al. (2013) applied self-organising neural networks with a Q-learning component for online generation of within-visual-range 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 rate after 120 episodes against a CGF with a fixed behaviour model. Furthermore, the network peaked at a 40% win rate against pilots in training, and below 10% against experienced pilots. Teng, Tan, Ong and Lee (2012) report using available air combat doctrine for building the state- and action-space for the Q-learning component by encoding expert knowledge as if-then rules.

Fourth, Liu and Ma (2017) applied deep reinforcement learning to generate air combat behaviour for an air combat agent. Deep reinforcement learning is a machine learning technique that combines (1) deep neural networks and (2) reinforcement learning. Deep neural networks are a class of neural networks that employ many layers of neurons (hence the term "deep"). The use of many layers allows the networks to not only learn (1) a mapping between input and output data, but also (2) their own feature detectors, by which the networks can adapt themselves to the most important features in the input data. The combination of a deep neural network with a reinforcement learning technique, such as Q-learning, results in a form of machine learning that is known as deep reinforcement learning. In deep reinforcement learning, the deep neural network is used to approximate the value function for the reinforcement learning technique. This way, the network can use its adaptive feature detectors to learn the values of state-action pairs.

In the past years, deep reinforcement learning has been shown to be a versatile and powerful technique. Recently, a deep reinforcement learning agent called ALPHAGO ZERO has learned to play the game of Go purely by self-play, and then continued to repeatedly defeat a previous version of itself (Silver et al., 2017b). The previous version, known as ALPHAGO LEE, had earlier received acclaim for defeating a human world champion (Silver et al., 2016). Liu and Ma (2017) tested their air combat agent with deep reinforcement learning against another agent that used a minimax decision making algorithm. The rewards for the learning agent were based on (1) the relative positioning of the agents, and (2) the optimal firing range of the learning agent's weapon. In an experiment consisting of 100 encounters the agent learned to defeat its opponent nearly 60% of the time. These encounters took place after a training session consisting of 5000 encounters.

In the literature, we see neural networks applied in two ways: (1) since the 1990s as a model for partial control of a CGF's behaviour, such as situation recognition (McMahon, 1990; Rodin and Amin, 1992), and recently (2) as a model controlling the entire behaviour of a CGF (Teng et al., 2013; Liu and Ma, 2017). Because of the black box nature of neural networks, controlling only a part of behaviour increases the possibilities of complete validation of the resulting models. Below we briefly discuss models for partial control. The approach of partial control of behaviour using neural networks is advocated by, e.g., Henninger, Gonzalez, Georgiopoulos and DeMara (2000). A recent example of neural networks learning only a specific part of CGF behaviour is the work by Kamrani et al. (2016), in which CGF representing soldiers learn a troop movement pattern. The use of neural networks for partial control allows for completing complex sub-tasks (e.g., situation classification instead of "air combat" as a whole task) while limiting the effects of the inaccessibility of trained networks, since only the networks performance on the sub-task has to be explained and validated.

2.4.2 Evolutionary algorithms

A second type of algorithm that has been applied to the generation of air combat behaviour is the evolutionary algorithm. The strength of evolutionary algorithms is the ability to generate and try multiple creative solutions simultaneously. However, as is the case with neural networks, their weaknesses are (1) the need for extensive learning phases, and (2) the inaccessibility of the resulting models (depending on the specific evolutionary technique that is used). Below we discuss five lines of development in which evolutionary algorithms were applied to air combat behaviour, viz. the works by (a) Mulgund, Harper and Krishnakumar (1998), (b) Smith, Dike, Mehra, Ravichandran and El-Fallah (2000a), (c) Kaneshige and Krishnakumar (2007), (d) Yao, Huang and Wang (2015), and (e) Koopmanschap, Hoogendoorn and Roessingh (2013).

Line (a): genetic algorithms. Mulgund et al. (1998) (continued by Mulgund, Harper and Zacharias, 2001) applied a genetic algorithm to find optimal formations for many-versus-many beyond-visual-range engagements. For this application, Mulgund et al. divided air combat tactics into three parts: (1) individual manoeuvres, (2) formations as a form of cooperation for small groups of aircraft, and (3) the use of individual manoeuvres and formations in large groups of aircraft. In their work, they focused on the latter as an optimisation problem, while using "conventional" individual tactics and small formations. Starting from a scenario with equal losses on both sides, the algorithm of Mulgund et al. was able to develop formations for as many as 16 aircraft. Using these large formations and the conventional tactics, all enemy CGFs were defeated without any defeats on the friendly side. While this is an impressive result, it only encompasses a small part of air combat behaviour generation. Furthermore, only a few parameters used by the algorithm are reported.

Line (b): learning classifier systems. Smith et al. (2000a) (see also the work by Smith, Dike, Ravichandran, El-Fallah and Mehra, 2000b) generated innovative one-versus-one within-visual-range behaviour for an experimental fighter jet using learning classifier systems (LCSs). The work by Smith and colleagues is a prime example of using evolutionary methods for the creative diversity of their solutions. It is explicitly stated that the goal of the study was the discovery of new behaviour, and not finding optimal behaviour. According to Smith et al., the automatic discovery of behaviour for a new aircraft allows simulation experts to give feedback to aircraft designers, customers, and operators about optimal ways to take advantage of the new aircraft's capabilities.

Line (c): artificial immune systems. An unconventional approach related to evolutionary algorithms was taken by Kaneshige and Krishnakumar (2007). The algorithm in this work was designed like an artificial immune system. The immune system selected manoeuvres (i.e., antibodies) to defeat detected intruders (i.e., antigens) in within-visual-range air combat. The parameters that were used were specifically chosen so that the algorithm was able to select manoeuvres within two seconds of calculation time. However, these parameters also appeared to limit the diversity of the solutions severely, as the algorithms quickly converged to the manoeuvres with the best performance.

Line (d): grammatical evolution. Yao et al. (2015) recently applied grammatical evolution to generate behaviour trees. At the core of their method was a genetic algorithm that operated on behaviour trees represented as bit-strings. The evolution of the behaviour trees was guided by a grammar. The grammar encoded three types of data: (1) the possible conditions and actions that the behaviour tree could use, (2) the parameters of these conditions and actions, and (3) the structure by which the conditions and actions could appear in the behaviour tree. Use of the grammar served two purposes: (1) it guided the evolution, limiting the search space, and (2) it kept the resulting behaviour tree accessible to humans. However, even though the grammar served to limit the search space, there is no mention of any constraints placed on the creativity of the genetic algorithm. Yao and colleagues tested their method in a one-versus-one beyond-visual-range air combat simulation. In the best case, the agent using the grammatical evolution method learned to outperform its opponent after 60,000 simulations.

Line (e): optimising a cognitive model. Koopmanschap et al. (2013) optimised a cognitive model for air combat CGFs by means of an evolutionary algorithm (see also Koopmanschap, Hoogendoorn and Roessingh, 2015). This cognitive model took the form of a network that connected observations to beliefs. Starting with the observations made by a CGF, the network enabled the CGF to form beliefs about its situation. The CGF then selected its actions based on its beliefs. The cognitive model allowed for the modelling of human-like features in the CGF's reasoning process, such as (1) *situation awareness*, (2) *surprise*, and (3) *theory of mind* (Merk, 2013). The network used by Koopmanschap et al. was constructed by a human expert with domain knowledge. The evolutionary algorithm was used to optimise the connection strengths between the observations and beliefs in the cognitive model. This way, the evolutionary algorithm was able to determine how strongly (combinations of) observations contributed to the formulation of specific beliefs. The network was tested in simulations with air combat CGFs. Koopmanschap et al. assigned a high fitness to the CGF if it defeated an opposing CGF, and a low fitness if the CGF took out a friendly CGF. The evolutionary algorithm outperformed both a hill climbing algorithm and a random search strategy. Wilcke, Hoogendoorn and Roessingh (2014) built upon the work by Koopmanschap et al. by letting the evolutionary algorithm determine the connections between the observations and beliefs, rather than only the connection strengths.

As can be seen above, evolutionary algorithms may provide creative and interesting solutions to complex problems. The creativity of evolutionary algorithms can be a great asset in developing behaviour. However, two drawbacks are: (1) the learning process takes time, and (2) the creativity of evolutionary algorithms must be guided, to ensure no loss of training control occurs.

2.5 Chapter summary

In this chapter, we have provided background information on four topics.

First, we took a detailed look at the four steps of the behaviour modelling process (see Section 2.1). The behaviour modelling process is the process by which behaviour models for training simulations are created today. It is a lengthy process with interdependent steps.

Second, we discussed the potential benefits and drawbacks of using machine learning in training simulations (see Section 2.2). The potential benefits are: (1) faster development of behaviour models compared to the behaviour modelling process, (2) the automatic detection of patterns in behaviour, and (3) online behaviour adaptation (see Subsection 2.2.1). The two potential drawbacks are: (1) emergence of unrealistic behaviour, and (2) the resulting loss of training control (see Subsection 2.2.2). In this thesis we focus on achieving the first potential benefit, and take care to avoid the two potential drawbacks.

Third, we introduced the three categories of machine learning tasks: (1) unsupervised learning tasks, (2) supervised learning tasks, and (3) reinforcement learning tasks (Section 2.3). Furthermore, we discussed the most important reinforcement learning concepts and reviewed the dynamic scripting reinforcement learning technique.

Finally, we reviewed past approaches to generating air combat behaviour models by means of machine learning (see Section 2.4). The two most commonly used techniques are (1) neural networks, and (2) evolutionary algorithms. However, these two techniques make it difficult to avoid the potential drawbacks of using machine learning in training simulations.

3 Team coordination

In this chapter we investigate research question 1: *To what extent can we generate air combat behaviour models that produce team coordination?*

Team coordination is an essential part of air combat. We will consider what team coordination means, by reviewing it from two perspectives: (1) the air combat perspective, and (2) the multi-agent system perspective. The field of multi-agent systems will provide us with a framework by which we can implement a variety of coordination methods. Subsequently we will experiment with them in our air combat simulations.

This chapter is structured as follows. First, we elaborate on the concept of team coordination from the two perspectives mentioned above (Section 3.1). Based on these perspectives, we develop and present three coordination methods: (1) TACIT, (2) CENT, and (3) DECENT. Then, we implement the three methods in a dynamic scripting environment (Section 3.2). Next, we determine the effect of the three coordination methods on the performance of a pair of CGFS by means of an experiment involving automated simulations (Section 3.3). We present the results of the experiment (Section 3.4) and then discuss them (Section 3.5). Finally, we summarise the chapter and answer research question 1 (Section 3.6).

This chapter is based on the following two publications.

- A. Toubman, J. J. Roessingh, P. Spronck, A. Plaat and H. J. Van den Herik (2014a). Dynamic Scripting with Team Coordination in Air Combat Simulation. In: *Modern Advances in Applied Intelligence: 27th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2014, Kaohsiung, Taiwan, June 3-6, 2014, Proceedings, Part I*. Ed. by M. Ali, J.-S. Pan, S.-M. Chen and M.-F. Horng. Vol. 8481. Lecture Notes in Computer Science. Kaohsiung, Taiwan: Springer International Publishing, pp. 440–449. ISBN: 978-3-319-07455-9. DOI: 10.1007/978-3-319-07455-9_46
- A. Toubman, J. J. Roessingh, P. Spronck, A. Plaat and H. J. Van den Herik (2014b). Centralized Versus Decentralized Team Coordination Using Dynamic Scripting. In: *Proceedings of the 28th European Simulation and Modelling Conference - ESM'2014*. Ed. by A. C. Brito, J. M. R. Tavares and C. Braganca de Oliveira. Porto, Portugal: Eurosis, pp. 129–134

3.1 Two perspectives on team coordination

In this section, we discuss the concept of team coordination. Team coordination is an extremely broad concept that can be viewed from many different perspectives. Therefore, we restrict ourselves to two relevant perspectives: (1) the air combat perspective and (2) the multi-agent system perspective. The air combat perspective shows us the kind of team coordination that we wish to integrate into the behaviour models of air combat CGFS (Subsection 3.1.1), whereas the multi-agent system perspective provides us with a framework to perform the task (Subsection 3.1.2). Combining the two perspectives will result in three coordination methods: (1) TACIT, (2) CENT, and (3) DECENT. We present the three methods in Subsection 3.1.3.

3.1.1 The air combat perspective

Today, the smallest unit that performs air combat missions is the *two-ship section* (also referred to by *two-ship* or *section*[1]). As the name implies, a two-ship consists of two aircraft. The two aircraft, as well as the pilots that fly them, are called the *lead* and the *wingman* (see, e.g., Borck, Karneeb, Alford and Aha, 2015). In general, the lead makes the tactical decisions for the two-ship, and the wingman follows and supports the lead. To succeed in their missions, the lead and the wingman in a two-ship need to carefully coordinate their actions.

To the best of our knowledge, the book *Fighter Tactics* by Shaw (1985) is the most comprehensive publicly available work on team coordination that takes place in a two-ship. However, the book is several decades old. It has missed substantial technological and doctrinal advances that have happened since its publication. Three of these advances are (1) the improvement of missile guidance and propulsion systems, (2) improved airborne radar technology, and (3) the shift from WVR to BVR air combat, made possible by (1) and (2) (cf. Bongers and Torres, 2014). Shaw (1985) largely focuses on WVR air combat, while BVR air combat is only mentioned in passing. However, the concepts presented by Shaw still surface regularly in modern sources, indicating that these concepts are also relevant today (cf. Bigelow, Taylor, Moore and Thomas, 2003; Aleshire, 2005; Crane, Bennett Jr, Borgvall and Waldelöf, 2006; Marken, Taylor, Ausink, Hanser and Anderegg, 2007; Hinman, Jahn and Jinnette, 2009; Laslie, 2015; Stillion, 2015).

According to Shaw (1985), team coordination in two-ships is based on the principle of *mutual support*. Shaw does not explicitly define the principle, but rather illustrates it with examples of behaviour. Here, we abstract Shaw's examples into two concepts: (1) creating situational awareness, and (2) a flexible division of roles. Below, we explain these two concepts.

The first concept, *creating situational awareness*, concerns the gathering of critical information, and using that information to form decisions. Shaw illustrates the importance of creating situational awareness with a straightforward example. A lone fighter aircraft has large blind spots behind and below the aircraft. Therefore, the pilot's ability to visually detect other aircraft

[1]In this thesis, we use the term *two-ship* to avoid confusion with other uses of the term *section*.

is limited. By having fighter aircraft fly in pairs, the blind spots of both aircraft are to a large extent reduced. Although visual detection has nowadays been largely replaced by radar, the same concept still applies. Two radars can be used to monitor a larger piece of airspace than is possible by using only one radar. Furthermore, the use of two radars allows, e.g., the tracking of specific target aircraft by one radar, while at the same time searching for additional aircraft using the second radar is possible. By having available more information that is relevant to their situation, the pilots can make more informed decisions on how to act.

The second concept, *a flexible division of roles*, partly follows from the first concept. The lead and the wingman begin their operations with a division of roles that is decided in advance. For instance, during the operations, when the lead of a two-ship is engaged by an opponent and has to perform defensive manoeuvres, the wingman remains free to observe the situation. As the lead is busy defending, the wingman is now in the better position to make tactical decisions for the whole two-ship. Therefore, rather than only observing, the wingman can (1) take over the tactical leadership of the lead, and (2) attack the opponent that is pursuing the lead. In other words, by properly coordinating, a two-ship is able to focus on offensive and defensive actions at the same time.

3.1.2 The multi-agent system perspective

The field of multi-agent systems is concerned with the operation of multiple actors in a single environment, such as CGFs in a simulation environment (see, e.g., Connors, Miller and Lunday, 2016). Team coordination has long been studied as an important part of multi-agent systems. Compared to a single agent, multi-agent systems have multiple benefits, such as (1) the ability to act in more than one physical location, (2) higher fault tolerance, as agents can take over the tasks of other agents in case of failure, and (3) flexibility, offered by the application of specialised agents that can cooperate in various manners (cf. Stone and Veloso, 2000; Yan, Jouandeau and Cherif, 2013; Ye, Zhang and Vasilakos, 2017). However, profitting substantially from these benefits requires that the agents coordinate their actions in some way.

Throughout the literature, coordination methods are classified along two axes: (1) the degree of centralisation of the coordination method, (2) the communication that takes place between the agents. Below, we discuss the two axes.

The first axis is the degree of centralisation of the coordination method. On one extreme of this axis are the methods in which the coordination of the agents is managed in a single, central agent. These methods are commonly called *centralised* coordination methods. In a multi-agent system with centralised coordination, the central agent receives information of all the other agents. It then uses this information to create a global "picture" of the environment. Based on this picture, the central agent plans (what it believes to be) the most optimal actions for each agent. The optimal actions are relayed to the other agents so that the actions can be executed (cf. McLennan, Molloy, Whittaker and Handmer, 2016).

On the other extreme of the same axis we see the methods in which the coordination is managed in a distributed fashion amongst the agents. We refer to these methods as *decentralised* coordination methods. Obviously, decentralised methods do not use a central agent to manage the coordination between agents. Rather, all of the agents in the system coordinate their actions amongst themselves. Often, only a local form of coordination is admitted or required (e.g., between agents working in the same room, or on the same task), as the actions of agents do not interfere with all other agents in the system (cf. Su, Zhang and Bai, 2016; Hou, Wei, Li, Huang and Ashley, 2017b).

The second axis is the communication between the agents. On one extreme of the axis we see agents that do not communicate at all. On the other extreme however, we do not see any well-defined actions either. The reason is that any communication that takes place between agents requires a communication scheme. Since all communication schemes involve a combination of (1) information that is communicated, and (2) a means of communication, a wide range of communication schemes is possible. For example, in terms of information, an agent might be able to communicate its observations to another agent, but not its intentions. At the same time, the available means could be noisy, time-lagged, or only allow a limited number of bytes to be transmitted. Coordination methods and agents that employ communication therefore have to be designed with the communication possibilities taken into account. Jennings, Sycara and Wooldridge (1998, p. 18) summarised the issue of designing communication schemes in a single question: "what and when to communicate?"

As has been implied in this section, there are multiple ways to design coordination methods. Designing a coordination method is one of the most challenging parts of inventing multi-agent systems (cf. Rodriguez-Aguilar, Sierra, Arcos, López-Sánchez and Rodriguez, 2015; Evertsz, Thangarajah and Papasimeon, 2017). The introduction of machine learning into the multi-agent system enables the designers of the system to take advantage of the creativity of machine learning in the design of the coordination method (cf. Tuyls and Weiss, 2012). Machine learning allows agents to develop a well-designed coordination method, and try to improve their operations while using that coordination method as a foundation (cf. Panait and Luke, 2005; Foerster, Assael, De Freitas and Whiteson, 2016; Havrylov and Titov, 2017). In the next section, we combine the two perspectives on team coordination into three coordination methods for CGFs.

3.1.3 Combining the perspectives into coordination methods

In this section, we combine (1) the air combat perspective on team coordination, with (2) the multi-agent system perspective on team coordination. Below, we first explain how we combine the two perspectives. Next, we present three coordination methods that follow from the combination.

From the air combat perspective, we derive the context of the agents for whom we are developing the coordination methods. The agents are (1) a lead CGF and (2) a wingman CGF, that together form a two-ship. Any coordination method for this two-ship should (1) help the

CGFs build situational awareness, and (2) enable a flexible division of roles.

From the multi-agent system perspective, we adopt the two axes along which coordination methods are classified: (1) the extent of the centralisation of the coordination among the agents, and (2) the extent of the communication among the agents. Crossing these two axes results in an axial system that is divided into four quadrants (see Figure 3.1). The four quadrants are:

1. The upper left quadrant, containing decentralised coordination methods without communication.
2. The lower left quadrant, containing centralised coordination methods without communication.
3. The lower right quadrant, containing centralised coordination methods with communication.
4. The upper right quadrant, containing decentralised coordination methods with communication.

Within these quadrants we are able to define particular coordination methods. We define three coordination methods: (1) a decentralised coordination method without communication called TACIT, (2) a centralised coordination method with communication called CENT, and (3) a decentralised coordination method with communication called DECENT. The lower left quadrant is left open without a coordination method. This quadrant requires a centralised coordination method without communication. However, without any form of communication taking place before or during encounters, it is quite difficult to envision what such a centralised coordination method would look like. We consider the development of a centralised coordination method without communication outside of the scope of our research. Below, we describe TACIT, CENT, and DECENT.

TACIT. The coordination method TACIT (see Figure 3.1, upper left) has the following two properties: (1) it is a decentralised method, viz. there is no central agent who controls all actions, and (2) it is a method without communication among the agents. This means that the lead and the wingman each select their own actions in an individualistic manner. They do not purposefully exchange information with each other (hence the term *tacit*). However, the lead and the wingman are able to observe (1) each other, and (2) the results of each other's actions, and then base the selection of their actions on these observations.

The use of TACIT provides an advantage to the designer of the behaviour of the lead and the wingman in the two-ship: (1) the behaviour of the lead can be designed with minimal regard for the wingman, and (2) vice versa (i.e., the behaviour of the wingman can be designed with minimal regard for the lead). However, conversely, the use of TACIT also makes it difficult to include explicitly coordinated interactions in the combined behaviour of the two-ship. Furthermore, it is possible that the behaviour of a team using TACIT will

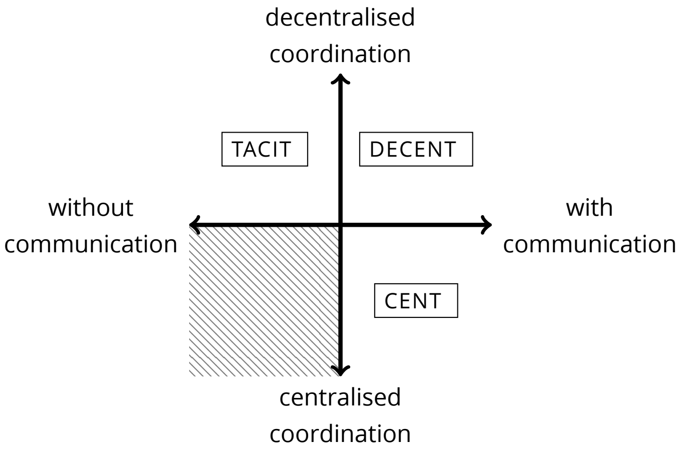


Figure 3.1 The two axes of team coordination: (1) the extent of the centralisation of the coordination (vertical axis), and (2) the extent of the communication between agents (horizontal axis). Placed on the axes are three coordination methods: (1) TACIT, the decentralised coordination method without communication (upper left), (2) CENT, the centralised coordination method with communication (lower right), and (3) DECENT, the decentralised coordination method with communication (upper right). We consider centralised coordination without communication (lower left) to be outside the scope of our research.

resemble least (out of the three coordination methods) the behaviour of a real-world two-ship, as both the lead and the wingman draw their own plan in the simulation.

CENT. The coordination method CENT (see Figure 3.1, lower right) has the following two properties: (1) it is a centralised method, viz. there is a central agent who coordinates all actions, and (2) it is a method with communication among the agents. We designate the lead as the central agent of the two-ship. This means that the lead selects all actions for both itself and the wingman. The actions selected by the lead are sent to the wingman to execute. The wingman helps the lead select actions by sending its observations to the lead. In other words, there is two-way communication between the lead and the wingman.

In CENT, the lead and the wingman are *tightly coupled*, as the lead needs to know the capabilities of the wingman in order to select actions for it. This presents a challenge to the designer of the behaviour of the two-ship. Furthermore, this challenge may grow when, in the future, a designer desires to reuse an existing lead CGF and wingman CGF to form a new two-ship using CENT. If the communication that occurs between the lead and the wingman is incompatible (viz. the lead does not know how to use the observations of the wingman to select actions for the it), the coordination will fail.

DECENT. The coordination method DECENT (see Figure 3.1, upper right) has the following two properties: (1) it is a decentralised method, and (2) it is a method with communication among the agents. By communicating, both the lead and the wingman receive information from each other's sensors, in addition to the information from their own sensors. The additional information leads to better situational awareness, by which both the lead and the wingman can make a better informed selection of actions.

In terms of design, DECENT resembles TACIT albeit with the ability of communication. Therefore, a two-ship that uses DECENT can in principle make a better informed decision (i.e., action selection) than a two-ship that uses TACIT. Furthermore, in contrast to CENT, the lead and the wingman perform their own action selection.

An important topic that we did not mention above is the topic of *learning*. By introducing machine learning to the coordination methods, the two-ship will be able to optimise their coordination with regards to achieving some goal (e.g., defeating an opponent). Machine learning affects the three coordination methods in the following manners. First, a lead and a wingman using TACIT will be able to learn how to act in each other's presence in order to achieve their common goal. Second, in the case of CENT, the central agent will be able to learn (1) how both (the agent itself and the other agent) should act, and (2) what information the central agent has to send to the other agent in order to start these actions. Third, when using DECENT, the lead and the wingman both have to learn individually (1) how to act, and (2) which information to send to each other in order to influence the other agent to act in a more desirable manner.

Returning to the air combat perspective on team coordination, we note that the three coordination methods and the introduction of machine learning now provide both (1) improved situational awareness, and (2) a flexible division of roles with the two-ship. By communicating information to each other (i.e., in the case of CENT and DECENT), the lead and the wingman can help each other build situational awareness. Furthermore, the introduction of machine learning into the coordination methods allows the lead and the wingman to learn how to (1) regulate and act themselves, and (2) coordinate with each other. Because the lead and the wingman share a common goal, we may assume that each of them will in a natural way learn to assume a particular role in the two-ship, such that each will (1) create tactical advantages for both themselves and the other, and (2) take advantage of the created tactical advantages in order to reach the common goal. In essence, the flexible division of roles is caused by the ability to learn.

3.2 Team coordination in dynamic scripting

In this section, we implement the three coordination methods (TACIT, CENT, and DECENT) in a dynamic scripting environment. This entails fitting the concepts of centralisation and communication into the rule-based framework as required by dynamic scripting. By means of the implemented coordination methods, a two-ship of CGFS will be able to learn how to coordinate their actions with each other. Additionally, for the two methods with communication (CENT and DECENT), the capacity to learn includes the ability to learn what and when to communicate.

We base the implementations in this section on the assumption that the coordinating two-ship will act as the red CGFS in the scenarios that are outlined in Appendix A.4. This means that the goal of the two-ship is to defeat one blue CGF (viz. hit blue with a missile), without being defeated themselves. Below, we describe the implementations of TACIT (Subsection 3.2.1), CENT (Subsection 3.2.2), and DECENT (Subsection 3.2.3) in detail.

3.2.1 Implementing TACIT

In order to implement TACIT, we translate its two main properties to a dynamic scripting environment. These properties are: (1) it is a decentralised method, and (2) it is a method without communication among the agents.

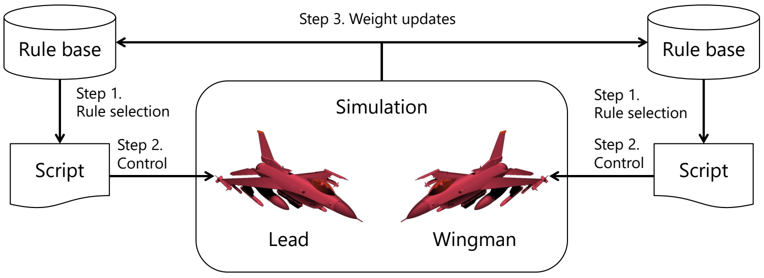
We translate the first property by letting the lead and the wingman in the two-ship be individual learners who each use their own rulebase. We treat the lead and the wingman as equals, in the sense that both receive the same rules in their rulebase (except for the wingman who has additional rules for formation flying, see later in this subsection). We translate the second property by designing the rules of each CGF in such a way that a rule only fires based on the observations made by the CGF to which the rule belongs (viz. no observations are communicated between the CGFS).

Figure 3.2a shows the learning process of a two-ship using TACIT. The lead (left) and the wingman (right) each have their own rulebase. The dynamic scripting algorithm generates a script from each of these rulebases (see Subsection 2.3.3, steps 1 to 3). The scripts are used to control the behaviour of the CGFS in the simulation. Based on the outcome of the simulation, dynamic scripting calculates the weight updates for the rules in the rulebases. During the simulations, there is no communication between the lead and the wingman (in contrast to e.g., DECENT, see Figure 3.2c). Below, we briefly describe the rulebases used by the lead and the wingman.

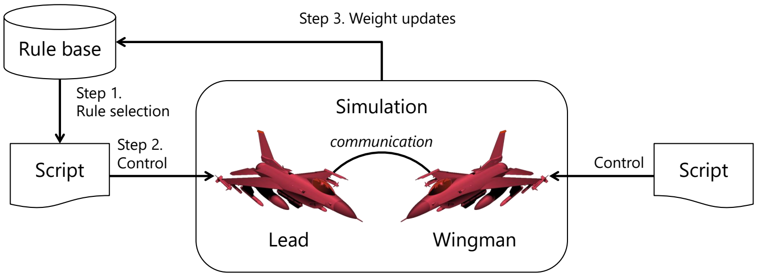
The two rulebases (one for the lead, one for the wingman) are presented in Appendix C. The rules in these rulebases are written in the LWACS scripting language that is described in Appendix B.1. Rather than discussing each individual rule, we divide the rules in each rulebase into three groups: (1) regular rules, (2) filler rules, and (3) default rules. Below, we describe the three groups of rules. At the end of the subsection, we briefly discuss the number of rules that are included in scripts when TACIT is used.

Regular rules. Regular rules are basic behaviour rules that map observations to actions. The lead has regular rules for four distinct types of behaviour: (1) firing missiles (five rules), (2) supporting[2] fired missiles (three rules), (3) evading incoming missiles (three rules), and

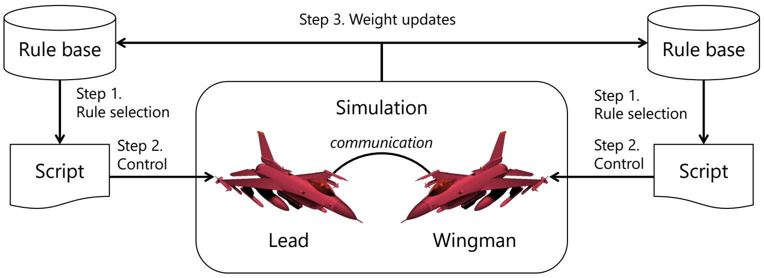
[2]Certain types of real-world missiles have to be *supported*, i.e., guided to the target by the radar of the aircraft that fired the missile. While the missiles in LWACS do not require being supported, we have included rules for this behaviour. The reason is that the behaviour (i.e., tracking the target with a radar and following its movements) may provide a



(a) TACIT. Both the lead and the wingman learn by means of dynamic scripting. There is no communication between them.



(b) CENT. Only the lead learns by means of dynamic scripting. The behaviour of the wingman is controlled by means of a predefined, non-learning script. There is two-way communication between the lead and the wingman.



(c) DECENT. Both the lead and the wingman learn by means of dynamic scripting. There is two-way communication between the lead and the wingman.

Figure 3.2 The three coordination methods implemented in dynamic scripting: (a) TACIT, (b) CENT, and (c) DECENT.

(4) evading opponents that have been detected using the radar warning receiver (RWR) (four rules). The wingman's regular rules include the same rules as the lead, plus rules for a fifth type of behaviour: flying in formation with the lead (five rules). For each type of behaviour, three to five variant rules are included, i.e., rules with slightly different values (e.g., firing from 50, 60, 70, 80, or 90 km).

Filler rules. Filler rules are rules that by design cannot fire, and therefore never execute an action. The purpose of the filler rules is to give the dynamic scripting algorithm the option to fill a script with rules, without forcing the algorithm to include rules in the script for the sake of reaching the required amount of rules. Without filler rules, the algorithm may include rules in scripts that have low weights (viz. they are "bad" rules). Despite their low weights, these rules can trigger during simulations and thereby cause the CGF to perform undesirable actions.

Because the filler rules cannot fire, they are no candidate for a weight increase or decrease when the CGF receives a reward or punishment, respectively. However, the weights of the filler rules are able to change by means of the weight redistribution performed by the dynamic scripting algorithm (see Subsection 2.3.3). In short, dynamic scripting keeps the sum of the weight values in the rulebase constant, by redistributing the weights of all rules whenever a change has to be made in the weight of any rule.

We include 11 filler rules in the rulebases of both the lead and the wingman. The first six filler rules are instrumental for dynamic scripting to fill an entire script with filler rules (because of the script size of six rules, see below). We add five additional filler rules to make the rulebases for TACIT and DECENT contain an equal number of rules. The only difference between TACIT and DECENT is the inclusion of rules for communication in the DECENT rulebases (see Figure 3.2c) Therefore, equalising the number of rules in the rulebases allows for a fair comparison of the learning speeds of the CGFs.

Default rules. Default rules are rules that provide fallback behaviour for the CGFs, when no other rules fire. These rules are automatically appended to every script that is generated by dynamic scripting. The purpose of the default rules is to aid the discovery of combinations of rules. For example, consider a generated script that contains a rule for supporting a missile, but does not contain a rule for firing a missile at any opponent. A CGF using this script will not defeat the opponent, since that requires a missile. By automatically appending a rule for firing a missile (even a sub-optimal rule), the CGF is given a chance to (1) defeat the opponent, (2) give a higher weight to the supporting rule, and (3) retry the supporting rule with a better firing rule from its own rulebase.

The rulebases of both the lead and the wingman include five default rules. Four of these five rules are shared by the lead and the wingman.

tactical advantage in some unexpected other manner.

1. a rule for returning the radar to search mode when the radar no longer detects any opponents,
2. a rule for tracking any opponents, when the radar detects them in search mode,
3. a rule for firing a missile when the lead has a firing opportunity within a range of 40 km,
4. a rule for supporting fired missiles,

lead a rule for flying in the general direction of the blue CGF (to ensure that contact is made between the red and blue teams),

wingman a rule for flying in formation with the lead.

The default rules have been assigned lower priority values (see Subsection 2.3.3) than the regular rules that provide the same behaviour, so that when such a regular rule is included in a script, the default rule never fires.

Script sizes using TACIT

Whenever the dynamic scripting algorithm generates a script from the rules in the rulebase, it only includes a preset number of rules in the script. We call this number the script size s. The script size is an important implementation detail, because it directly affects the complexity of possible scripts, and thus also the complexity of the behaviour of the agents that use the scripts. However, there are no guidelines for selecting a correct script size.

We define the script size of the scripts that dynamic scripting generates for CGFs that use TACIT to be $s = 6$. We choose this script size because it allows scripts to include at least one regular rule for each distinct type of behaviour (see above, *Regular rules*).

Using s and the size of a rulebase, we are able to calculate the number of possible scripts S. The rulebase of the lead contains 26 rules, whereas the rulebase of the wingman contains 31 rules. Here, we do not consider the five default rules that are appended to every generated script. Since a rule can only be included once in the same script, we arrive at the following number of possible scripts (1) S_{lead} for the lead, and (2) $S_{wingman}$ for the wingman.

$$S_{lead} = \binom{26}{6} = 230\,230$$

$$S_{wingman} = \binom{31}{6} = 736\,281$$

The total number of combinations of scripts that the two-ship can use in a simulation is therefore $230\,230 * 736\,281 = 169\,513\,974\,630$. This number indicates the size of the combined search space that the dynamic scripting algorithms (one for the lead, one for the wingman) will have to traverse, in order to find the optimal behaviour of the two-ship (with optimal behaviour seen from the perspective of the reward function, see Subsection 3.3.2). Note that the total number of combinations of scripts does not equal the total number of different behaviours that the two-ship can display in simulations, because of (a) duplicated rules in the rulebases (e.g., the filler rules) and (b) rules that supersede each other (e.g., firing from at most 80 km also implies firing from 50 km if the opportunity presents itself). An exploratory calculation involving (a) six rules that provide the same behaviour and (b) five rules that supersede each other results in 41,719 possible behaviours for the lead and 192,160 possible behaviours for the wingman. Using these numbers, the total number of possible behaviours for the two-ship is $41\,719 * 192\,160 = 8\,016\,723\,040$.

3.2.2 Implementing CENT

In order to implement CENT, we translate its two main properties to a dynamic scripting environment. These properties are: (1) it is a centralised method, and (2) it is a method with communication among the agents.

We translate the first property by designating the lead as the central agent in the two-ship. The lead learns by means of dynamic scripting. Its rulebase contains rules that (1) define behaviour for itself, and (2) coordinate with the wingman. The wingman can be controlled by any non-learning control method. To stay within the rule-based paradigm, we control the wingman by means of a predefined script. We translate the second property by allowing the CGFS to send messages to each other. The act of sending a message is implemented as an action that can be part of the consequence of a rule. This way, messages can be sent conditionally (i.e., only when a rule with such an action fires). Below, we first describe (A) the rulebase of the lead. Next, we describe (B) the script of the wingman. At the end of the subsection, we briefly discuss the number of rules that are included in scripts when CENT is used.

A: The rulebase of the lead

The rulebase of the lead contains rules that define (1) behaviour for the lead and (2) coordination with the wingman. Regarding the behaviour of the lead, the rulebase includes the same three groups of rules that were defined for the TACIT lead: (1) regular rules, (2) filler rules, and (3) default rules. The only difference is the number of filler rules. The CENT lead's rulebase has six filler rules, rather than the 11 filler rules in the rulebase of the TACIT lead (see Subsection 3.2.1).

For the coordination with the wingman, the CENT lead's rulebase contains two additional groups of rules: (4) rules for formation flying (five rules), and (5) directive rules (13 rules). Below, we describe these two groups.

Rules for formation flying. The rulebase of the lead includes five rules for formation flying. These rules are the same as the five regular rules for formation flying which the TACIT wingman has in its rulebase (see Subsection 3.2.1). However, when one of these rules fires, the lead communicates to the wingman that it should assume a certain formation, rather than assuming the formation itself.

Directive rules. Directive rules fire on specific messages that the lead receives from the wingman. We identify the following eight messages that the wingman can send to the lead.

1. The wingman observes a new opponent by means of its radar.
2. The wingman observes a new opponent by means of its RWR.
3. The wingman observes that a missile is flying towards the wingman.
4. The wingman is able to fire a missile at an opponent from <50 km away.
5. The wingman is able to fire a missile at an opponent from <60 km away.
6. The wingman is able to fire a missile at an opponent from <70 km away.
7. The wingman is able to fire a missile at an opponent from <80 km away.
8. The wingman is able to fire a missile at an opponent from <90 km away.

Upon firing, each directive rule causes the lead to send an instruction to the wingman. In response to each of the first three messages, the lead instructs the wingman to perform one of three actions.

1. Turn right ninety degrees,
2. turn left ninety degrees, or
3. turn towards the observed opponent (in case of message 1 or 2).

In response to the remaining five messages, the lead instructs the wingman to fire. Note that the lead does not respond to the messages the same way in each simulation (or at all), since the directive rules are subject to dynamic scripting's rule selection.

B: The script of the wingman

The wingman uses a predefined script to control its behaviour. Consequently, all of the rules in the script are able to fire in each simulation. The script of the wingman consists of three groups of rules: (1) informative rules, (2) executive rules, and (3) default rules. Below, we describe the three groups of rules.

Informative rules. Informative rules send a message to the lead to inform it of a new observation made by the wingman. The script includes eight informative rules. Five of the informative rules inform the lead that the wingman is able to fire at the opponent from within a certain distance (50, 60, 70, 80, or 90 km). The remaining three informative rules inform the lead of one of three events.

1. The wingman has detected a new opponent by means of its radar,
2. the wingman has detected a new opponent by means of its RWR, or

3. the wingman has detected that a new missile has been fired at it (see A, *Directive rule*).

Executive rules. Executive rules cause the wingman to execute an action upon the reception of an instruction to do so from the lead. The script includes 14 executive rules. Five of the executive rules are for flying in formation (see A, *Rules for formation flying*). Five executive rules are for firing at the opponent (from 50, 60, 70, 80, and 90 km). Two executive rules are for executing evasive manoeuvres (turning right ninety degrees and turning left ninety degrees, respectively). The remaining two executive rules direct the wingman towards the observed position of the opponent.

Default rules. The script of the wingman includes four default rules. Four of these five rules are shared by the lead and the wingman.

1. A rule for returning the radar to search mode when the radar no longer detects any opponents,
2. a rule for tracking any opponents, when the radar detects them in search mode,
3. a rule for firing a missile when the wingman has a firing opportunity within 40 km, and
4. a rule for flying in the general direction of the blue CGF.

Apart from the default rules, the wingman has no capacity to select and execute actions by itself. All actions of the wingman are selected by the lead, and then communicated to the wingman by means of a directive rule. Because the script of the wingman is predefined and does not change between or during simulations, all instructions that are received lead to the firing of the corresponding executive rule.

Script sizes using CENT

We define the script size of the scripts that dynamic scripting generates for the lead CGF to be $s = 12$. This is a larger script size than the script size we defined for CGFs that use TACIT. The reason for the larger script size is that the scripts that are generated for the lead have to include both (1) rules for the lead's behaviour, and (2) rule for the wingman's behaviour.

The rulebase of the lead contains 39 rules, excluding the default rules. Therefore, we arrive at the following S_{lead}.

$$S_{lead} = \binom{39}{12} = 3\,910\,797\,436$$

Since the script of the wingman is predefined (i.e., $S_{wingman} = 1$), the number of possible scripts for the lead is also the number of possible behaviours that the two-ship can display in simulations (viz. the search space for behaviours). This search space is two orders of magnitude smaller than the search space for TACIT (see Subsection 3.2.1). This suggests that it should be easier for CENT CGFS to find good solutions (viz. faster learning). However, in CENT, only the weights of the rules in one rulebase will be optimised (i.e., for the lead), compared to the two rulebases in TACIT (i.e., for the lead and the wingman). We will look at the specific effects of the centralised method and the two decentralised methods in the experiment that is presented later (see Section 3.3).

3.2.3 Implementing DECENT

In order to implement DECENT, we translate its two main properties to a dynamic scripting environment. These properties are: (1) it is a decentralised method, and (2) it is a method with communication among the agents.

DECENT shares its first property with TACIT, and its second property with CENT. We therefore turn to TACIT and CENT for translating the two properties. First, as in TACIT, we translate the first property by letting the lead and the wingman in the two-ship be individual learners which each use their own rulebase. Second, as in CENT, we translate the second property by allowing the CGFS to send messages to each other, by means of the rules in the rulebases.

Figure 3.2c shows the learning process of a two-ship using DECENT. The lead (left) and the wingman (right) each have their own rulebase. The dynamic scripting algorithm generates a script from each of these rulebases (see Subsection 2.3.3, steps 1 to 3). The scripts are used to control the behaviour of the CGFS in the simulation. Based on the outcome of the simulation, dynamic scripting calculates the weight updates for the rules in the rulebases. During the simulations, there is communication between the lead and the wingman.

We base the DECENT rulebases on the rulebases that are used by the TACIT lead and wingman. We make two changes to these rulebases. The first change is that we add broadcast actions to the rules in the rulebases. The broadcast actions allow the CGFS to communicate to each other what actions they each are performing. The second change is that we replace five of the filler rules by response rules, i.e., rules that only perform actions upon the reception of specific broadcasts. Below, we describe the two changes. At the end of the subsection, we briefly discuss the number of rules that are included in scripts when DECENT is used.

Broadcast actions. We append a broadcast action to each of the regular rules and the default rules (except for default rule 5 of both the lead and the wingman, see Subsection 3.2.1). The appended action sends the *intention* of the rule that fires to the other CGF in the two-ship. We call the appended action the broadcast action because it indiscriminately sends a message to the other CGF, with no regard for whether (1) the sending CGF or (2) the receiving CGF expects the message to be useful. It is up to the rules in the script of the receiving CGF to take action (or not) upon receiving the message (see below, *Response rules*).

As an example of the appended broadcast action, each rule that makes the CGF fire a missile now also sends the message "firing at opponent" to the other CGF. In total, we identify seven intentions: (1) searching for an opponent, (2) tracking an opponent, (3) firing at an opponent, (4) supporting a fired missile, (5) engaging an opponent detecting by means of RWR, (6) evading an opponent detected by means of RWR, and (7) evading a detected incoming missile. The use of broadcast actions to send their intentions to each other gives the CGFS (1) a coarse way to inspect each other's internal state and (2) the means of adjusting their own actions to those of the other CGF in the two-ship.

Response rules. During the design of the response rules, we noticed that it was a difficult task to produce meaningful rules that respond to each of the different broadcasts. Therefore, we only included five response rules. The five response rules are as follows.

1. If the other red CGF is evading an opponent or a missile from that opponent, head towards the approximate location of the opponent.
2. If the other red CGF is tracking the opponent, firing a missile at the opponent, or supporting a missile, then head towards the location of the opponent.
3. If the other red CGF is evading an opponent, also make an evasive manoeuvre by turning 180 degrees.
4. Equal to response rule (3) but turn ninety degrees right.
5. Equal to response rule (3) but turn ninety degrees left.

Script sizes using DECENT

For DECENT, we use the same script size as for TACIT ($s = 6$). Furthermore, the sizes of the rulebases of the lead and the wingman are also equal to those for TACIT (26 and 31 rules, respectively). As a result, the S_{lead} and $S_{wingman}$ for DECENT are equal to those for TACIT as well.

3.3 Experimental setup

We design an experiment to determine which of the three coordination methods leads to the most effective behaviour (i.e, leads to the highest amount of scenarios won). The experiment consists of automated simulations. In this section, we present the setup of the experiment. The setup is divided into six parts: a description of LWACS (Subsection 3.3.1), the red team (Subsection 3.3.2), the blue team (Subsection 3.3.3), the scenarios that were used (Subsection 3.3.4), the independent and dependent variables (Subsection 3.3.5), and a description of our method of analysis and the criteria for comparison (Subsection 3.3.6).

3.3.1 The Lightweight Air Combat Simulator

The simulations in this experiment are performed in the Lightweight Air Combat Simulator (LWACS) simulation program. This program was developed at the Netherlands Aerospace Centre (NLR) for the goal of straightforwardly evaluating behaviour models. Because it was developed at Netherlands Aerospace Centre (NLR), we had complete access to the source code and were able to modify the program to fulfil our simulation needs. The LWACS program is explained in detail in Appendix A. Below, we provide a brief summary of its functionality.

We set up LWACS to simulate an empty section of airspace. This section of airspace is inhabited by fighter jet CGFS. Each fighter jet CGF carries three kinds of devices: (1) a radar by which it can detect other aircraft, (2) four air-to-air missiles that are used for disabling opponent CGFS, and (3) a RWR by which it can detect radar emissions of other aircraft. The physics model used by LWACS is based on classical mechanics, i.e., the CGFS are able to move through the section of airspace without any notion of aerodynamics. Furthermore, we restricted the movement of the CGFS to the horizontal plane in order to simplify the rules that were required for the behaviour of the CGFS. This way, we were able to get a clear first indication of the suitability for dynamic scripting in the BVR air combat domain.

The section of airspace that is simulated in LWACS contains a particular element of randomness which makes the simulations non-deterministic. This element is the *probability-of-kill* of missiles. When a missile hits its target, the probability of the missile disabling its target is calculated based on the distance the missile has flown from the moment of launch. As a result, no two simulated encounters are the same, even though the CGFS in the encounters may use the same behaviour model. To draw conclusions on the effectiveness of the behaviour of the CGFS, we simulate a large number of encounters in the experiment.

In LWACS, behaviour models can be created by means of a scripting language. We present this language in Appendix B. The language is used to write scripts containing behaviour rules. In each encounter that is simulated in LWACS, a script is assigned to each CGF. During the encounter, the script governs the behaviour of the CGF. The presence of existing code for parsing and executing scripts made it simple to integrate the dynamic scripting algorithm directly into LWACS.

3.3.2 Red team

The *red team* (also referred to as *red*) consists of two fighter jet CGFS, a lead and a wingman. The capabilities of the CGFS are described in Appendix A. The goal of the red team is to learn how to defeat the blue team in three different scenarios (see Subsection 3.3.4). The red team uses each of the three coordination methods implemented in dynamic scripting: (1) TACIT, (2) CENT, and (3) DECENT.

The dynamic scripting algorithm requires a reward function to calculate the adjustment of the weights in rulebases. We design a simple reward function, that we call BIN-REWARD. BIN-REWARD stands for *bin*ary *reward*. The reward function provides a reward signal to each CGF with a value of 1 if the CGF defeats its opponent, and a value of 0 if a member of its own team (including itself) is defeated. The same reward signal is provided to both the lead and the wingman after a simulation. The lead and the wingman then use the reward signal to update the weights in their own rulebases (see Figure 3.2, *Step 3*). We will delve more deeply into the subject of reward functions in Chapter 4.

3.3.3 Blue team

The *blue team* (also referred to as *blue*) consists of a single fighter jet CGF. The capabilities of the CGF are described in Appendix A.2. The goal of the blue team is to defeat the red team, by hitting one of the reds with a missile. The behaviour of blue CGF is governed by scripts (see Appendix A.3).

The use of a team with a single CGF may surprise the reader, as this chapter emphasises the importance of teamwork in air combat. There are three reasons for including only one CGF in the blue team. First, even if teamwork is important, a lone fighter jet armed with air-to-air missiles is still a dangerous opponent. We stress the danger of this opponent by declaring the blue team the winner of a scenario when the blue team hits only one of the red CGFs. Second, at this point, it is unclear how well the red team will perform when encountering any opposing agents, as red's rulebases (and script) are newly designed. The use of a single blue agent is a starting point for evaluating the performance of red. Third, as stated in this chapter and the previous chapter, it is difficult to manually design the behaviour for one agent, let alone two. As a result, the more agents with manually designed behaviour we add, the less confident we can be that together the agents represent a team capable of providing adequate opposition. Later in the thesis, we introduce blue teams with more than one CGF (see Chapter 5 and Chapter 7).

3.3.4 Scenarios

Simulators need to be configured so that a specific situation or event is displayed in the simulation that is run. This configuration is commonly called the scenario (see, e.g., Durak, Topçu, Siegfried and Oğuztüzün, 2014). We define the term scenario below.

Definition 3.1 (Scenario). A scenario is a simulator configuration that defines (1) the number of CGFs in the simulation, (2) the teams that the CGFs belong to, (3) the initial positions, headings, and speeds of the CGFs, and (4) the termination criteria of the simulation.

In the automated simulations that are performed in LWACS, we use the four two-versus-one scenarios described in Appendix A.4: (1) the basic scenario, (2) the close range scenario, (3) the evasive scenario, and (4) the mixed scenario. These four scenarios are repeatedly presented to red. This way, red is able learn how to behave in each of the four scenarios over a run of encounters.

3.3.5 Independent and dependent variables

The experiment uses two independent variables: (1) the three coordination methods (TACIT, CENT, and DECENT), and (2) the four scenarios (basic, close range, evasive, and mixed). The combination of these independent variables results in a 3×4 fully factorial design with twelve conditions. In each condition, we obtain the win rates of red. The win rates are the dependent variable in the experiment.

3.3.6 Method of analysis

We make use of two measures to analyse the win rates of red: we calculate (1) the final performance, and (2) the turning points. Below, we briefly explain the two measures.

Measure 1: Final performance. The win rates of red are expected to show a learning curve, viz. increasing in the beginning of a run of encounters, then levelling off when the scripts generated for red cannot be optimised further. We are interested in measuring the performance of red after it has levelled off, i.e., after red has learned to defeat blue by means of dynamic scripting. We define the final performance of red as red's win rates averaged over the last fifty encounters in each run. The final performance indicates in which condition the reds find the most effective behaviour against blue.

Measure 2: Turning points. We calculate the turning point for each condition. This measure was first presented by Spronck et al. (2006). A sliding window ($n = 10$) is placed over the encounters in each run. The turning point is the last encounter in the sliding window, when the red team has won more encounters in that window than the blue team. We use the turning point as an indication of the learning speed of red.

Because of the use of a sliding window, the turning point measure experiences both a floor effect and a ceiling effect. The floor value of this measure is the nth encounter, which is the case when the turning point is found in the first n encounters. The ceiling value of the turning point measure is encounter n_{max}, where n_{max} is the number of encounters in the run. Even if no turning point exists in the results of a run, a turning point at encounter n_{max} will be measured. The floor and ceiling effects should be taken into account when interpreting the results of the turning point measure.

We investigate the results from the two measures by means of analyses of variance (ANOVAS). Using the ANOVAS, we determine whether red's final performance and/or turning points differ between the three coordination methods and/or four scenarios.

3.4 Experimental results

In this section, we present the results of the experiment.[3] For each condition, we simulated 50 runs of encounters, with each run consisting of 150 encounters. In total, we simulated 90,000 encounters for the experiment.

Figure 3.3 shows the win rates of red against blue. The win rates are divided over the four scenarios. For each scenario, the win rates of red using each of the three coordination methods are shown. The final performance and the turning points of red are shown in Table 3.1 and Table 3.2, respectively. Two two-way ANOVAs were performed.

First two-way ANOVA (final performance)

The first two-way ANOVA was conducted on the influence of the two independent variables (coordination method, scenario) on the final performance of red. All effects were statistically significant at the $\alpha = .05$ level. The main effect of coordination method yielded an F ratio of $F(2, 588) = 16.353$, $p < .001$, indicating a significant difference between the final performances of red using each of the three coordination methods. The main effect of scenario yielded an F ratio of $F(3, 588) = 52.689$, $p < .001$, indicating a significant difference in the final performance of red in each of the four scenarios. Furthermore, the interaction between the coordination methods and the scenarios was significant, $F(6, 588) = 3.844$, $p < .01$. We performed a post hoc Tukey honest significant difference (HSD) test (see, e.g., Holmes, Moody, Dine and Trueman, 2016) to determine the significant differences in final performance between specific pairs of (a) coordination methods and (b) scenarios. First, the post hoc test revealed that (a) the final performance of the reds using CENT differed significantly from that of the reds using either TACIT or DECENT, $p < .001$. Consequently, the final performance of the reds that used either TACIT or DECENT did not differ significantly. Second, the post hoc test revealed that (b) the final performance differed significantly between all scenarios, $p < .001$, except between the close range and mixed scenarios.

Second two-way ANOVA (turning points)

The second two-way ANOVA was conducted on the influence of the two independent variables (coordination method, scenario) on the turning points. The main effect of coordination method was not found to be statistically significant. However, the main effect of scenario was statistically significant at the $\alpha = .05$ level. The main effect of scenario yielded an F ratio of $F(3, 588) = 21.957$, $p < .001$, indicating a significant difference between the turning points of red in the four scenarios. We performed a post hoc Tukey HSD test to determine the significant differences

[3] The results that are reported in this thesis differ from the results that were reported in the two publications. Since these publications, LWACS received multiple updates. The results in this thesis are from new simulations that were run using the latest version of LWACS.

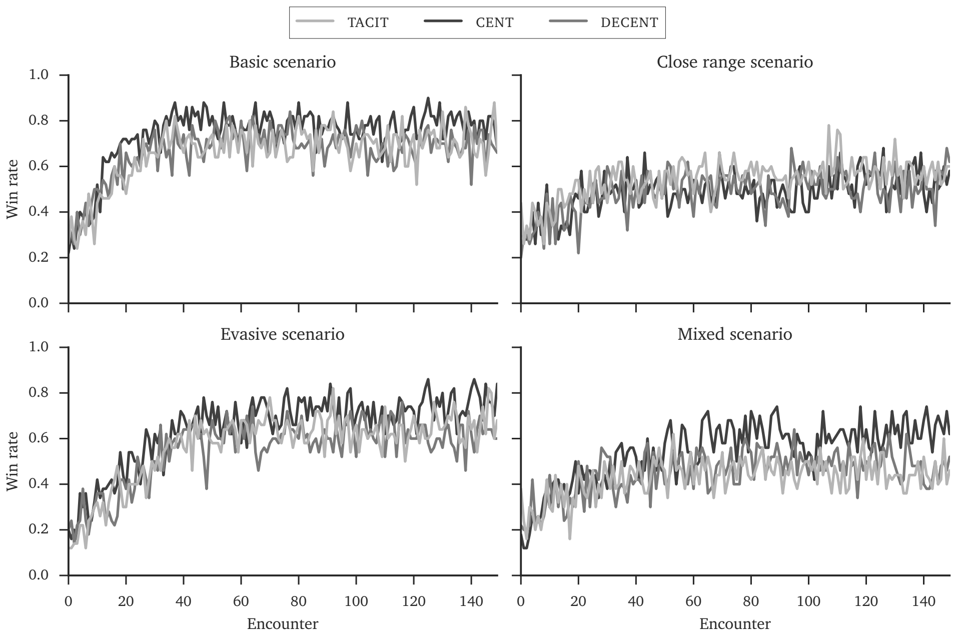


Figure 3.3 The win rates of red against blue. Red used one of three coordination methods: (1) TACIT, (2) CENT, and (3) DECENT. The behaviour of blue depended on four scenarios: (1) the basic scenario, (2) the close range scenario, (3) the evasive scenario, and (4) the mixed scenario.

Table 3.1 The final performance of red. A higher final performance is better.

	Basic scenario		Close range scenario		Evasive scenario		Mixed scenario		Grand mean	
Coordination method	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
TACIT	.704	.213	.584	.187	.643	.190	.452	.117	.596	.202
CENT	.778	.184	.535	.119	.736	.155	.619	.180	.667	.187
DECENT	.686	.161	.521	.149	.621	.157	.486	.117	.579	.166

Table 3.2 The turning points of red. A lower turning point is better.

	Basic scenario		Close range scenario		Evasive scenario		Mixed scenario		Grand mean	
Coordination method	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
TACIT	26.9	13.7	26.9	13.1	36.0	17.4	45.8	30.8	33.9	21.4
CENT	21.4	9.4	33.8	23.9	31.3	20.0	39.8	28.7	31.6	22.5
DECENT	23.2	11.0	40.1	27.4	38.2	15.4	43.5	22.5	36.2	21.4

in turning points between specific pairs of scenarios. The post hoc test revealed that the turning points differed significantly between all scenarios, $p < .01$, except between the close range and evasive scenarios.

3.5 Discussion

In this section, we discuss the results of our experiment. We partition the discussion into four parts: our key finding (Subsection 3.5.1), the effect of centralised coordination on performance (Subsection 3.5.2), the learning process of coordinating CGFS (Subsection 3.5.3), and the way forward (Subsection 3.5.4).

3.5.1 Key finding

Our key finding is that the results suggest that out of the three coordination methods that we tested, the CENT method leads to the most effective behaviour for the red two-ship. In three of the four scenarios (viz. the basic, evasive, and mixed scenarios), the CENT method clearly outperforms both TACIT and DECENT. In these three scenarios, CENT achieves the highest final performance and the lowest turning points, meaning that it is able to find the best rules at the earliest point in the learning process.

3.5.2 The effect of centralised coordination on performance

What sets CENT apart from TACIT and DECENT is that in the CENT method, there is only one CGF (i.e., the lead) that learns by optimising its rulebase. The wingman of this CGF is controlled by means of a non-changing script. In contrast, in both the TACIT and DECENT methods, both the lead and the wingman are each optimising their own rulebases at the same time. Therefore, it appears that the presence of only one rulebase that needs to be optimised is a benefit to the performance of the red two-ship. Since each of the two red CGFS is dependent on the other for winning each encounter (viz. defeating the blue and getting a reward), it takes longer for the CGFS to select rules that complement the rules that are selected by their teammate.

It is only in the close range scenario that the performance of CENT is overtaken by another method. In the close range scenario, it is the TACIT method that reaches both the highest final performance and the lowest turning point. Based on the results, it appears that in this scenario, it is both (a) the decentralisation of the coordination, and (b) the lack of communication that provide an advantage in this scenario. The blue in the close range scenario is scripted to only fire at the reds when it has approached them within a range of 50 km. This leaves the red CGF under fire with little time to effectively evade the missile. Furthermore, while its teammate is being fired upon, the remaining red only has the same amount of time to setup a counterattack while the blue is busy performing its own attack. The data suggests that in tight situations like this, it is marginally better to forego the communication, and therefore for each red CGF to try to directly take advantage of the situation.

3.5.3 The learning process of coordinating CGFs

To illustrate the learning process of a CGF, we plot the weights of the rules in a rulebase as they change over time. We do so for two rulebases: the rulebase of the red lead using the CENT method in the basic scenario (see Figure 3.4), and the rulebase of the red lead using the CENT method in the mixed scenario (see Figure 3.5). The figures show the progression of the weights of each rule (column) from the first encounter (top row) to the last encounter (bottom row). A darker colour indicates a higher weight. In the figures, the rules are identified by their internal code names for brevity (see the bottom of the figures). The specific rules that are associated with these code names can be found in Appendix C.

We make three observations in Figure 3.4 and Figure 3.5. The first observation is that in both figures, two rules come out with the highest average weights. These rules are (1) `evadeMissile180`, the code name for the rule "if I detect a missile flying at me, turn 180 degrees", and (2) `wingmanAskFireFrom90`, the code name for the rule "if my wingman says it is able to fire from 90 km, tell it to do so". The red lead discovers that these two rules are favourable at an early stage in the learning process.

The second observation is the light patch in Figure 3.4, from code name `fireFrom50` to `support-right`. These rules relate to the firing of missiles. The light patch indicate that the red lead does not favour firing missiles itself, but rather leaves its wingman to fire the missiles to defeat the blue. This is supported by the first observation.

The third observation is that the lead is quite indecisive on the best behaviour for the wingman. This can be seen in both Figure 3.4 and Figure 3.5 in the columns to the right of `wingmanAskFireFrom90`. In both figures, these columns are lightly coloured, meaning that the rules that these columns represent received weight increases in some of the runs over which we averaged, and no weight increases in other runs. While there is seemingly no pattern to the colours in these columns, we can distinguish a small difference. In Figure 3.4, the colours appear smoothed, whereas in Figure 3.5, they appear spotty and irregular. From the smoothed colours we conclude that in different runs of encounters, the rulebase of the lead has converged to different combinations of rules for the behaviour of the wingman. In contrast, the irregular colours show how in the mixed scenario, the lead keeps trying out different combinations of rules for the behaviour of the wingman because the blue opponent continually changes its behaviour. We made similar observations in the rulebases that were used with TACIT and DECENT. As the two figures show, the weights in the rulebases reflect (a) the nature of the scenarios that the rulebases were used in, as well as indicate (b) the flexibility of dynamic scripting, seen from the different combinations of rules that lead to clear divisions in roles between the CGFs.

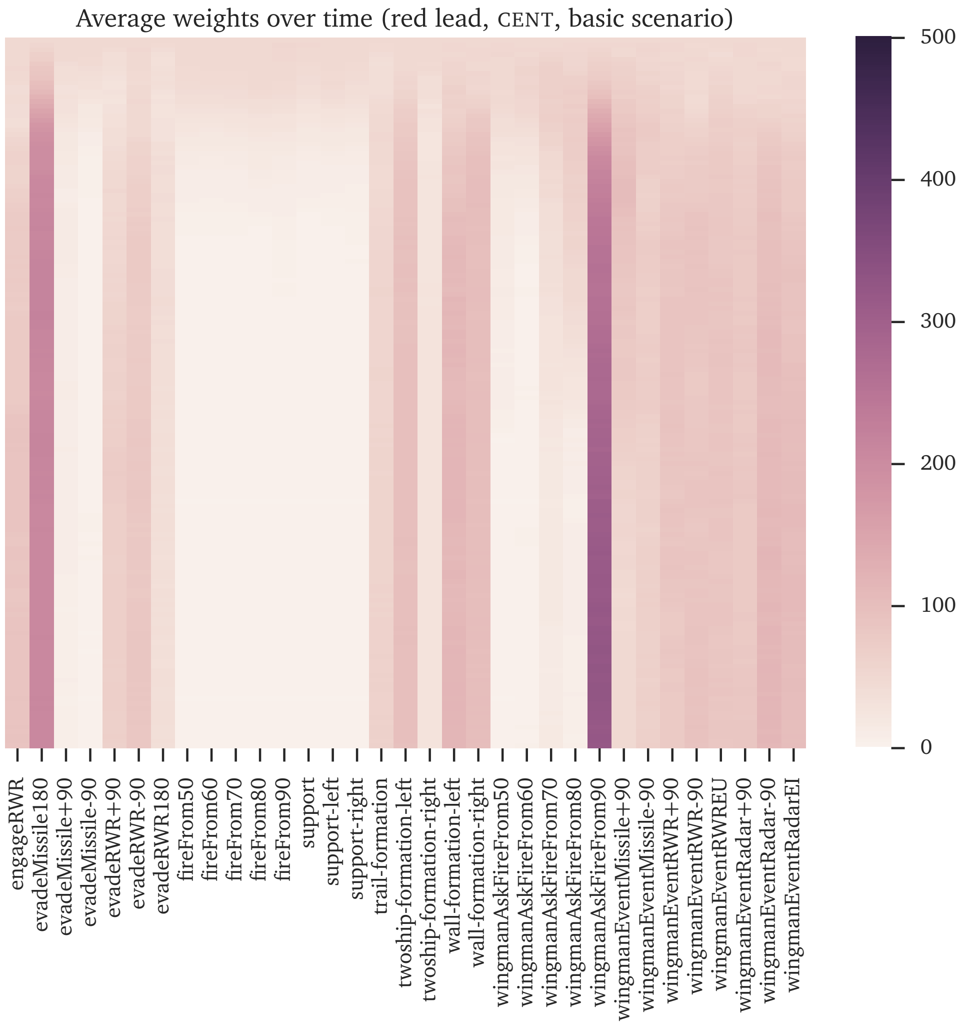


Figure 3.4 The weights of the rules in a rulebase shown as a heatmap graph. The graph shows the progression of the weights (columns) over time (rows, 150 encounters from top to bottom). The colour indicates the value of the weights (see legend on the right). This graph shows the weights in the rulebase of the red lead that used the CENT method in the basic scenario. The weights are averaged over 50 runs of encounters.

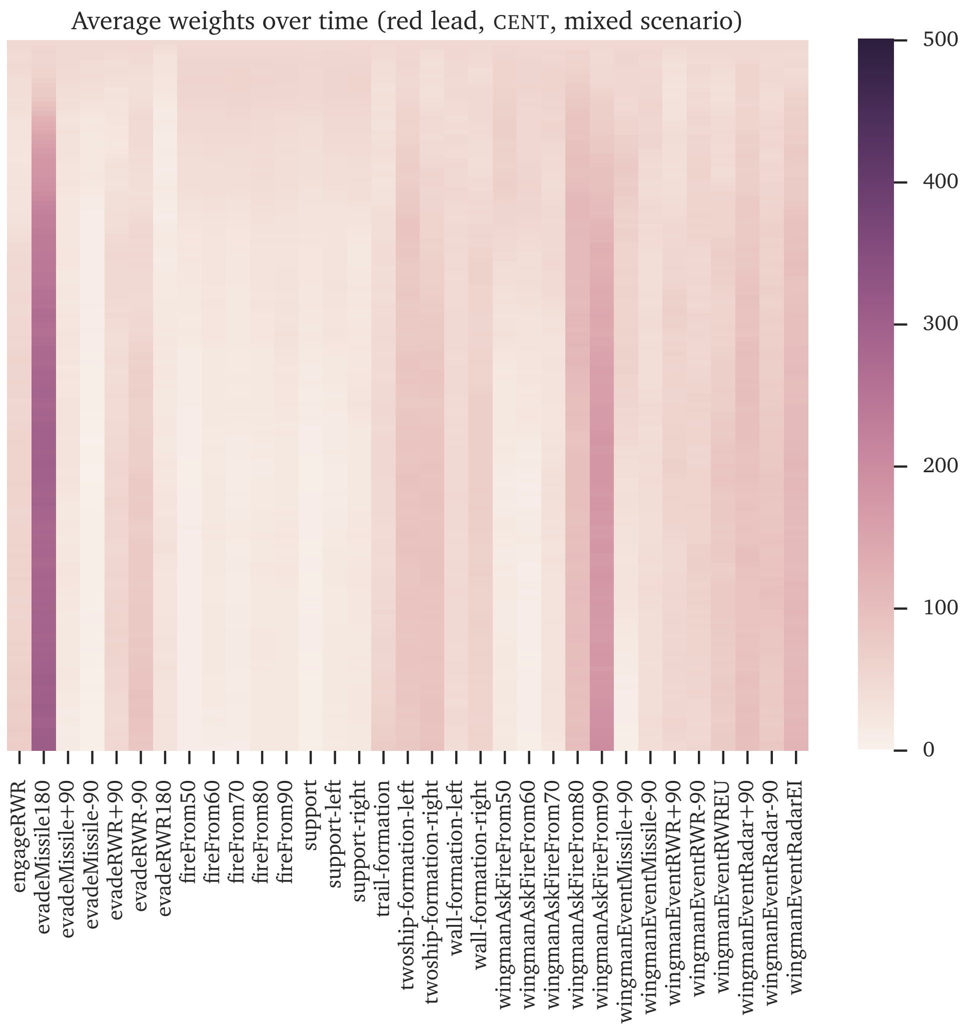


Figure 3.5 The weights of the rules in a rulebase shown as a heatmap graph. The graph shows the progression of the weights (columns) over time (rows, 150 encounters from top to bottom). The colour indicates the value of the weights (see legend on the right). This graph shows the weights in the rulebase of the red lead that used the CENT method in the mixed scenario. The weights are averaged over 50 runs of encounters.

3.5.4 The way forward

Although our results favour the CENT method, we still use a different coordination method in the research that we perform in the remainder of this thesis. The situation is as follows. In Chapters 4 through 7, we use the DECENT method for coordination between the CGFS. We have two reasons for this choice.

First, the use of DECENT is somewhat historical. In early experiments, DECENT provided the best performance. Based on this outcome, the choice was made to use DECENT in our simulations to answer the remaining research questions. However, the outcome presented in this chapter is the result of new simulations (based on the idea that new developments should be tested always as much as possible). Hence, we used the latest version of LWACS, as well as revised rules in the rulebases. Although the outcome surprised us, we decided to continue our research with the methodology chosen (see Section 3.3). Still, we report the current results extensively, since they are candidates for future research in this context.

Second, in our view, DECENT has the most interesting properties out of the three coordination methods presented in this chapter. These properties are (a) the communication between the CGFS, and (b) the fact that both CGFS are trying to optimise their own rulebases at the same time. The two properties open the door for CGFS that (a) have to learn to communicate and cooperate with each other, and (b) each CGF may perhaps use a different machine learning technique. Therefore, out of the three coordination methods, we have taken DECENT as the most attractive method for the research in the remainder of this thesis.

3.6 Answering research question 1

In this chapter, we investigated the generation of behaviour models that allow for team coordination between CGFS. Specifically, we addressed research question 1. Below, we summarise the work done in this chapter, and how this work answers the research question.

Research question 1 reads: *To what extent can we generate air combat behaviour models that produce team coordination?* To answer this question, we reviewed the subject of team coordination from two relevant perspectives: (1) the perspective of air combat, and (2) the perspective of multi-agent systems. Combining these perspectives resulted in three coordination methods (Section 3.1), each of which we implemented in a dynamic scripting environment (Section 3.2): (1) the decentralised coordination method without communication TACIT, (2) the centralised coordination method with communication CENT, and (3) the decentralised coordination method with communication DECENT. The coordination offered by these methods was completely performed within the rule-based framework of dynamic scripting.

In conclusion, we answer research question 1 as follows. The dynamic scripting environment is flexible. It allows us to implement and experiment with three coordination methods: TACIT, CENT, and DECENT. We have shown that each of the three methods enables the CGFS to learn effective team behaviour in automated simulations. Therefore, the answer to the research question is that we are fully able to generate air combat behaviour models that produce team coordination by means of dynamic scripting. However, out of the three methods, the CENT method clearly provides the best performance. This is apparent in three out of the four scenarios that we used in the automated simulations. Especially in the mixed scenario, the CGFS using the CENT method demonstrate their ability to quickly adapt to the blue's changing behaviour. For this reason, we recommend to first consider the use of centralised coordination in future air combat simulations. Thereafter the scientific comparison between CENT and DECENT can be continued.

4 Improving the reward function

In this chapter, we investigate research question 2. This research question reads: *To what extent can we improve the reward function for air combat CGFs?*

We begin this chapter by introducing reward functions and their role in reinforcement learning (Section 4.1). For learning complex tasks by means of reinforcement learning, it is common to use a simple reward function that only rewards the agent when a particular task is completed (see, e.g., Večerík, Hester, Scholz, Wang, Pietquin et al., 2017). Designing reward functions is the topic of section 4.2. We discuss three types of designing reward functions: manual design, inverse reinforcement learning, and inverse reward design. The BIN-REWARD reward function (see Chapter 3) is an example of designing a straightforward reward function. However, its application leads to two problems: sparse rewards and unstable rewards. The first problem, *sparse rewards* (Section 4.3) means that the agent has to try many (combinations of) actions before a reward is obtained and the task-completing behaviour is reinforced. We propose a solution to sparse rewards in the form of the new DOMAIN-REWARD reward function. The second problem is *unstable rewards* (Section 4.4), meaning that the environment is non-deterministic and the same actions can lead to different results. We propose a solution to unstable rewards in the form of the new AA-REWARD reward function.

Moreover, we provide an overview that includes (a) a comparison of the properties and (b) a formal description of each of the BIN-REWARD, DOMAIN-REWARD, and AA-REWARD (Section 4.5). We compare the effect of the three reward functions on the performance of agents that learn to control air combat CGFs by means of an experiment (Section 4.6). We present the results of the experiment (Section 4.7) and discuss them (Section 4.8). Finally, we conclude the chapter by answering research question 2 (Section 4.9).

This chapter is based on the following publication.

- A. Toubman, J. J. Roessingh, P. Spronck, A. Plaat and H. J. Van den Herik (2015a). Rewarding Air Combat Behavior in Training Simulations. In: *Systems, Man, and Cybernetics (SMC), 2015 IEEE International Conference on*. Hong Kong: IEEE Press, pp. 1397–1402. DOI: `10.1109/SMC.2015.248`

4.1 Reward functions in reinforcement learning

The reward function plays a central role in reinforcement learning (cf. Alpaydin, 2010; Nowé and Brys, 2016; Sutton and Barto, 2018). It provides the feedback by which reinforcement learning agents learn to behave. In the beginning of an agent's learning process, the agent performs actions at will. These actions each cause some change in the environment, viz. they change the environment's *state*. The more this change is a desirable change in the environment (see below), the more reward is provided to the agent. When and how much the agent is rewarded is determined by the reward function.

The goal of a reinforcement learning agent is to collect as much reward as possible. Rewards are used by the agent to reinforce the behaviour that caused the agent to receive the rewards, so that the same behaviour will be repeated in the future. As a result, the reward function acts as the agent's teacher or critic. The reward function tells the agent what is desirable behaviour, defined below.

Definition 4.1 (Desirable behaviour). For a reinforcement learning agent, desirable behaviour is behaviour that is beneficial to achieve the intended goal of the agent.

For instance, in an air combat simulation, the behaviour of (an agent that controls a) CGF that ultimately provides the most training value for the trainees, may be the most desirable behaviour. However, in, e.g., a video game, behaviour that provides the most entertainment value for the player of the game is the most desirable. Precisely this sort of knowledge has to be captured in reward functions, so that the agent learns which behaviour is desired.

The remainder of this section is outlined as follows. First, we provide a formal description of reinforcement learning to show how reward functions influence the learning process of agents (Subsection 4.1.1). Next, we briefly describe how rewards influence the learning process of agents that learn by means of dynamic scripting (Subsection 4.1.2).

4.1.1 A formal description of reinforcement learning

In this subsection, we provide a formal description of reinforcement learning. In particular, we describe the workings of the Q-learning algorithm, which is currently one of the most commonly applied reinforcement learning algorithms (originally introduced by Watkins and Dayan, 1992, and restated by Mnih, Kavukcuoglu, Silver, Rusu, Veness et al., 2015; Nowé and Brys, 2016; Sutton and Barto, 2018). In Q-learning, reinforcement learning concepts are clearly represented, allowing a straightforward discussion of the components (e.g., agents, actions, and rewards) that make up the algorithm. In Subsection 4.1.2, we discuss how these components fit into dynamic scripting. Furthermore, Q-learning has served as the foundation for modern reinforcement learning algorithms such as *deep Q-learning* (Silver et al., 2016).

We describe the agent as being in an environment with state s_t at time step t, where the state belongs to the set of all possible states $s_t \in S$. At each time step, the agent performs one of its actions $a_t \in A$. When the agent performs its action a_t in s_t, the following occurs: (a) the time step increases to $t+1$, (b) the agent receives reward r_{t+1}, and (c) the environment changes state to s_{t+1}. The value of r_{t+1} is calculated by a reward function $r(s_t, a_t)$.

The agent's choice of action in each state is determined by the agent's policy $\pi : S \rightarrow A$. The policy maps each state $s \in S$ to the action $a \in A$ that the agent should perform in that state. The goal of the agent is to find the optimal policy π^* that maximises the amount of reward that the agent expects to receive based on previously executed policies.

Given the state s_t and the policy π, the agent can calculate the *value* $V^\pi(s_t)$ of the policy, viz. the *expected cumulative reward* obtained by following policy π starting in state s_t. The calculation of $V^\pi(s_t)$ is shown by Equation (4.1).

$$V^\pi(s_t) = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \tag{4.1}$$

Here γ, which is called the *discount factor*, controls the short- or farsightedness of the agent. If $\gamma \rightarrow 0$, the agent is only interested in immediate rewards, whereas if $\gamma \rightarrow 1$, the agent considers all rewards that can be obtained in the infinite future (viz. assuming that the task has no predefined duration).

Using Equation (4.1), we can accurately formulate the learning task of the agent. The agent needs to find the optimal policy π^*, which is the policy that maximises $V^\pi(s)$ for all states. The definition of π^* is shown by Equation (4.2).

$$\pi^* = \operatorname*{argmax}_\pi V^\pi(s), \forall s \tag{4.2}$$

Equation (4.2) shows that the optimal policy is the policy with the highest expected cumulative reward, regardless of which states the agent encounters. Because the agent needs to perform actions to change the state of the environment, and thereby receive more reward, we rewrite the definition of π^* as follows. In all states, the optimal action to take in state s_t is the action that directly leads to the most reward, plus the value of the next state of the environment. This is shown by Equation (4.3).

$$\pi^*(s_t) = \operatorname*{argmax}_a \left[r_{t+1} + \gamma V^\pi(s_{t+1}) \right] \tag{4.3}$$

Because the agent starts learning in a new, unknown environment, it does not perfectly know $V^\pi(s_t)$ from the beginning. Therefore, the agent has to *interact* with its environment, to determine and record which actions in which states lead to which rewards. To this end, the agent maintains the *action-value function* $Q(s_t, a_t)$. The action-value of a state-action pair is defined as the immediate reward obtained by performing a_t in s_t, plus the discounted value of following the policy in the new state s_{t+1}. This is shown by Equation (4.4). The problem of finding the optimal policy can now be rewritten as taking the actions that have the highest action-value in each state. This is shown by Equation (4.5).

$$Q(s_t, a_t) = r_{t+1} + \gamma V^{\pi^*}(s_{t+1}) \tag{4.4}$$

$$\pi^*(s_t) = \operatorname*{argmax}_{a_t} Q(s_t, a_t) \tag{4.5}$$

The action selection performed in Equation (4.5) is *exploitative*, viz. it always selects the optimal action with the highest Q-value. It is possible that different actions will lead to even higher Q-values. However, as long as the agent does not select these different actions, it will not discover their Q-values. Therefore, the agent should sometimes *explore* by performing a sub-optimal action. A common method of doing this is by introducing a so-called ϵ-greedy mechanism. By such a mechanism, the agent selects a random action with a probability equal to the ϵ parameter. Otherwise, the agent selects the optimal action. This is shown by Equation (4.6).

$$a_t = \begin{cases} \pi^*(s_t) & \text{with probability } 1-\epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases} \tag{4.6}$$

By supplying the agent with an *update rule*, the agent is able to adjust the action-values of actions as new rewards are obtained. The update rule that is used by Q-learning is shown by Equation (4.7).

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \eta\Big[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\Big] \tag{4.7}$$

Here, η is the *learning rate*. This is a parameter that scales how strongly behaviour should be reinforced by newly received rewards. The exact determination of r_{t+1} for each state transition is left to the designer of the reinforcement learning system (viz. the combination of the learning agent and its environment) to implement in the reward function.

4.1.2 The role of rewards in dynamic scripting

In dynamic scripting, rewards play the same role as in other reinforcement learning techniques such as Q-learning (Watkins and Dayan, 1992). The goal of the agent is to search for an optimal policy, thereby maximising the expected cumulative reward. However, in dynamic scripting, the mechanism by which rewards affect the search for the optimal policy is slightly different from that for, e.g., Q-learning.

In Q-learning, a reward can be obtained after each action that is performed by the agent. The reward is then immediately used to update the Q-value of this action, as shown in Equation (4.7). The updated Q-value changes what the agent believes to be the optimal policy, as shown in Equation (4.5). Therefore, after each reward, the agent learns and implements better behaviour.

In dynamic scripting, the rewards received by the learning agent are used to change the weights of the rules in the agent's rulebase. As in Q-learning, dynamic scripting rewards each state-action pair. In dynamic scripting, the possible state-action pairs are the rules. However, whereas in Q-learning rewards can be provided to the agent after each action, this is not the case in dynamic scripting. Rather, rewards are only provided in the terminal state of each encounter. We define the terminal state below.

Definition 4.2 (Terminal state)**.** The terminal state r_{terminal} is the last state in an encounter.

Rewards that are only provided in the terminal state are known as sparse rewards, which we discuss in Section 4.3. We refer to reward functions that only specify a fixed reward for a single terminal state, and no rewards (i.e., a reward of 0) for all other states, as *binary reward functions*. In the case of sparse rewards, each reward indicates to the agent how well the agent's entire script (the equivalent of a policy in Q-learning) is performing, rather than a specific rule that encodes a state-action pair.

Upon receiving a reward, the necessary changes to the weights in the rulebase are calculated by means of the *adjustment function*. The adjustment function that is used in this thesis is shown in Equation (4.8) (see also Toubman et al., 2014a). The function assumes that the reward r lies within the range $[0, 1]$. The resulting weight changes based upon r are restricted to the range $[-25, 50]$.

$$\text{adjustment}(r) = \max((r - 0.5) * 100, -25) \tag{4.8}$$

By use of this adjustment function, if the agent's behaviour was desirable and resulted in a reward > 0.5, the weights of the contributing rules are increased. Conversely, if the agent's behaviour was undesirable and resulted in a small reward < 0.5, the weights of the contributing rules are decreased. In the case of a reward of 0.5, no weight adjustment takes place.

Based on early trials, we restricted the maximum decrease in weight to -25, instead of using the symmetrical range $[-50, 50]$. This restriction had an important effect on rules that most of the time lead to successful behaviour, but that would sometimes still receive a punishment. The effect was that the punishment would not decrease the weight of the rule by so much (relatively to the other rules in the rulebase) that it would no longer be eligible for selection the next time a script was generated. The non-deterministic effect of actions leads to a problem known as unstable rewards, which we further discuss in Section 4.4.

4.2 Designing reward functions

In this section, we discuss how reward functions are designed. The goal of designing reward functions is to create a reward function that allows the learning agent to learn the desired behaviour as efficiently as possible. We assume that for each task, there is an *optimal* reward function. Below, we define the optimal reward function.

Definition 4.3 (Optimal reward function)**.** The optimal reward function is the function that allows the learning agent to discover the optimal policy in the minimal number of time steps.

We refer to reward functions that approach the optimal reward function as *good* reward functions. According to Nowé and Brys (2016), “[d]efining a reward function requires some experience, however coming up with a reward function is often quite straightforward.” While we agree that it is often straightforward to come up with *some* reward function, it requires more than “some experience” to define a *good* reward function. The design of good reward functions is quite difficult, as for each learning task, the designer has to consider *when*, *what*, and *how much* to reward (Janssen and Gray, 2012; Hadfield-Menell, Milli, Abbeel, Russell and Dragan, 2017). Sutton and Barto (2018) have listed the design of reward functions as one of the important frontiers in reinforcement learning.

The work by Chrabaszcz, Loshchilov and Hutter (2018) illustrates well how a “straightforward” reward function may influence the search for the optimal policy. They applied an evolutionary strategy algorithm to allow an agent to learn policies for playing a selection of Atari games. As rewards, the agent was given the in-game scores. Therefore, desirable behaviour was implicitly defined as all behaviour that maximised the scores in the games. In one particular game, the agent discovered two distinct bugs that caused the agent to obtain an infinitely high score. One of these bugs was previously unknown. By solely focusing on the scores, in some ways the agent learned to circumvent the game, rather than play it. In real-world applications, such behaviour may present safety concerns.

Ratner, Hadfield-Menell and Dragan (2018) identify three methods by which reward functions are designed: (a) manual design of reward functions, (b) inverse reinforcement learning, and (c) inverse reward design. We review the three methods below.

(a) Manual design of reward functions. A reinforcement learning expert specifies desirable states and the associated reward signals.

(b) Inverse reinforcement learning. A method called inverse reinforcement learning (cf. Abbeel, Coates, Quigley and Ng, 2007; Kitazato and Arai, 2018) is the inference of a reward function by observing behaviour that is demonstrated by an agent other than the learning agent. In other words, an agent (e.g., a human expert) demonstrates how a task should be performed, and the learning agent has to discover the goal (e.g., some terminal state) that this other agent is trying to reach. However, the risk exists that the learning agent infers some goal that is different from the true desired goal. This is especially true in real-world tasks, as shown by, e.g., Abbeel et al. (2007).

(c) Inverse reward design. Inverse reward design is a recent method in which a given reward function is treated as an observation that approximates the intended reward function (Hadfield-Menell et al., 2017). The observed reward function is interpreted as belonging to its context, viz. to the environment in which the learning agent is situated during learning.

By the use of inverse reward design, the agent is able to reason about the uncertain effectiveness of its reward function in newly encountered environments. Inverse reward design is most helpful in situations where the training environment may differ strongly from the operational environment of the learning agent. For instance, this method allows a learning self-driving car to detect new, unseen situations, and then automatically adopt a more conservative policy to avoid accidents.

In Chapter 3, we made use of a manually designed binary reward function called BIN-REWARD. This function provided to the learning agents a reward signal of one for winning a scenario (viz. defeating the opponent), and a reward signal of zero for not winning a scenario. The reward signal was provided to the agents after the terminal state had been reached in each encounter. The evaluation of intermittent states in air combat is a long-standing problem and a recurring topic of research (cf. Schreiber, Schroeder and Bennett Jr., 2011; Ömer and Ayan, 2013; Ximeng, Rennong and Ying, 2018). So far, no straightforward solution exists. The same problem occurs in, e.g., the game of Go. For instance, the well-known Go-playing programs ALPHAGO (Silver et al., 2016) and ALPHAZERO (Silver et al., 2017b) were also provided binary reward signals ($+1$ for winning a game, -1 for losing a game).

Binary rewards provide the learning agent with information that is certain: winning an encounter or game is desirable, and losing is not. However, the learning agent has to adjust its entire policy on the basis of this limited information. The use of binary rewards leads to two problems. The first problem is *sparse* rewards. Especially when learning a new task, the agent will need to randomly explore the space of states and actions before the first reinforcement of desirable behaviour can take place. The second problem is *unstable* rewards. This problem occurs in environments that are not completely deterministic, such as air combat. In these environments, because of non-deterministic events, performing a_t in s_t may lead to (a) different s_{t+1} each time that a_t is performed in s_t, and therefore also to (b) different r_{t+1} for the same (s_t, a_t) pair. In the two sections that follow, we further discuss sparse rewards (Section 4.3) and unstable rewards (Section 4.4) and propose a solution to each of the two problems.

Looking ahead to Section 4.3 and Section 4.4, we see that previous research has offered solutions to the problems of sparse and unstable rewards in two forms: (1) new reward functions, and (2) structural changes to the learning algorithms. In our own search for a satisfying solution to these problems, we are mainly interested in how much the performance of learning agents can be improved by formulation of a better reward function. The reason is two-fold. First, the structural changes that are proposed are specific to the learning techniques (e.g., Q-learning). Modifying dynamic scripting may inadvertently alter the properties that made it suitable for our research (e.g., the properties of computational speed and variety of behaviour). Second, the design of reward functions for agents learning air combat behaviour may also be relevant to research on the evaluation of human fighter pilots (cf. Schreiber et al., 2011; MacMillan, Entin, Morley and Bennett Jr, 2013; Ömer and Ayan, 2013; Petty and Barbosa, 2016; Ximeng et al., 2018).

4.3 Sparse rewards

In this section we discuss the problem of sparse rewards. The section is outlined as follows. We begin by describing the problem (Subsection 4.3.1). Next, we discuss the *reward shaping* technique (see, e.g., Grześ, 2017) which is commonly used to counteract sparse rewards (Subsection 4.3.2). We continue by providing an overview of how sparse rewards are dealt with in the literature (Subsection 4.3.3). Finally, we propose our own solution to sparse rewards in the form of the DOMAIN-REWARD reward function (Subsection 4.3.4).

4.3.1 Problem description

In descriptions of reinforcement learning (such as in Subsection 4.1.1), it is assumed that the learning agent receives a reward after each action that it has performed. However, when a binary reward function is used, the agent only receives a reward after the terminal state. If the agent does not receive a reward after each action, the rewards are called *sparse rewards* (cf. Petrik and Scherrer, 2009; Večerík et al., 2017). The opposite of sparse rewards are *dense rewards*.

The problem that sparse rewards present to reinforcement learning agents is that the agents have to spend a rather long time to explore a large variety of actions (viz. trying out random actions in states) before a reward is received. It is only at the moment when the reward is received that the agent can reinforce the behaviour which leads to that reward. This is known as sample inefficiency (cf. Goyal, Brakel, Fedus, Lillicrap, Levine et al., 2018; Kaushik, Chatzilygeroudis and Mouret, 2018). Because of sample inefficiency, a large number of interactions is needed to find an optimal policy.

4.3.2 Reward shaping

A common technique for counteracting sparse rewards and increasing sample efficiency is reward shaping (see, e.g., Grześ, 2017). Reward shaping is the augmentation of the reward function with heuristics. These heuristics are extra rewards for reaching certain intermediate states. The intermediate states and the associated rewards are chosen by the designer of the reward function based on domain knowledge. In other words, the designer provides the learning agent with "stepping stones" that the agent can use to understand what behaviour is desired from it.

While reward shaping has lead to successes (see Subsection 4.3.3), it is not without risk. If the designer specifies the wrong intermediate states, it is possible for the states to cause locally optimal policies, viz. policies that obtain the rewards for the intermediate states, but that also block advances towards a globally optimal policy. An example is provided by Popov, Heess, Lillicrap, Hafner, Barth-Maron et al. (2017), who taught a robotic hand the task of stacking blocks. As part of their research, Popov et al. shaped the reward function so that an intermediate reward would be given for grasping the block. In one case, they observed that the robot had learned to grasp a block. However, the manner in which the robot had grasped the block made it impossible to correctly stack the block on top of another block. This shows that even though reward shaping may sometimes seem like a natural manner of providing more rewards, the effectiveness greatly depends on how well the intermediate states that are rewarded for are defined.

4.3.3 Sparse rewards in the literature

Below, we provide six examples of solutions to sparse rewards that have been presented in the literature. We divide the examples into (1) solutions in the air combat domain in the form of specific reward functions, and (2) domain-independent solutions. In the air combat domain, we review the work by (1a) Ma, Ma and Song (2014), (1b) Yao et al. (2015), and (1c) Leuenberger and Wiering (2018). Furthermore, three domain-independent methods were recently proposed for dealing with sparse rewards. These methods are (2a) hindsight experience replay by Andrychowicz, Wolski, Ray, Schneider, Fong et al. (2017), (2b) backward learning by Edwards, Downs and Davidson (2018), and (2c) non-uniform action-value initialisation by Sutton and Barto (2018).

(1a) Air combat solution by Ma et al. (2014)

Ma et al. defined a continuous reward function for agents learning to control air combat CGFs. This reward function provides a reward at each time step. The reward is based on the geometry between the CGFs and the opposing CGFs. The function is shown in Equation (4.9).

$$r_t = \left[\frac{(1-|\mathrm{AA}_t|/\pi)+(1-|\mathrm{ATA}_t|/\pi)}{2}\right]\exp\left(-\frac{|R_t|-M_t}{\pi k}\right) \tag{4.9}$$

The reward function uses five parameters:

1. AA_t, the aspect angle[1] between the learning CGF and its opponent at time t,
2. ATA_t, the antenna train angle[2] between the learning CGF and its opponent at time t,
3. R_t, the range between the learning CGF and its opponent at time t,
4. M_t, the expect maximum effective range of a missile at time t,
5. and k, a weighting factor for the influence of R_t. The value of k that is used for learning is not mentioned.

[1]The aspect angle is "[the] relative angle between the longitudinal symmetry axis (to the tail direction) of the target [aircraft] and the connecting line from target [aircraft]'s tail to attacking [aircraft]'s nose" (Ma et al., 2014). An aspect angle of 0 indicates that the attacking plane can shoot straight at the target plane's tail, without risk of being shot back by the target plane.

[2]The antenna train angle is "the angle between attacking [aircraft]'s longitudinal symmetry axis and its radar's line of sight" (Ma et al., 2014). Modern on-board radar systems can adjust their heading independently of the aircraft to some extent. An antenna train angle of 0 indicates that the radar is aligned with the longitudinal axis, and therefore has its maximal room to manoeuvre in all directions. This increases the probability of the attacker maintaining a radar lock on the target in the near future.

According to Ma et al. (2014), the rewards provided by their reward function increase from the worst state (viz. the learning agent is under attack from directly behind) to the optimal state (viz. the learning agent attacks the target from directly behind) both continuously and monotonously. However, depending on the discount rate used, this reward function might preclude the exploration of policies that set up a near-optimal state by first moving to a sub-optimal state. For instance, it might be desirable to set up a missile shot by first performing some defensive manoeuvre. It is not mentioned what discount rate is used.

(1b) Air combat solution by Yao et al. (2015)

Yao et al. defined a function by which they measured the fitness of evolved air combat behaviour models. Technically, fitness functions serve a purpose slightly different from the one served by reward functions. In evolutionary algorithms, the fitness function is indirectly the optimisation target of the learning algorithm: viz. evolved solutions are selected based on their fitness, although the solutions are evolved to complete a task, and not to receive a high fitness *per se*. In contrast, in reinforcement learning, the learning algorithm directly optimises policies for obtaining the highest reward. However, the formulation of the function by Yao et al. makes it suitable as a reward function as well, which is why we have included it here. The fitness function is shown in Equation (4.10).

$$r_{\text{terminal}} = w_1 * r_{\text{outcome}} + w_2 * r_{\text{safe}} + w_3 * r_{\text{missile}} \tag{4.10}$$

The fitness function uses three weighted parameters:

1. r_{outcome}, which is one for winning, zero for losing, and one-third for a draw,
2. r_{safe}, which is the ratio of time spent safely during the encounter (viz. not tracked by a radar or a missile) to the duration of the encounter,
3. and r_{missile}, the ratio of missiles that were fired by the agent *and* that hit their target, to the total number of missiles fired by the agent.

Yao et al. (2015) used $w_1 = 0.7$, $w_2 = 0.2$, and $w_3 = 0.1$ in their experiment, thereby placing most emphasis on winning, but they also somewhat reward doing it safely and efficiently.

(1c) Air combat solution by Leuenberger and Wiering (2018)

Leuenberger and Wiering used a shaped reward function to reward their agent in a WVR air-to-air combat simulation. The simulated environment incorporated simple models of gravity and aerodynamics. Their learning agent controlled a CGF equipped with an on-board cannon. The cannon fired bullets in a straight line. The goal of the agent was to destroy an opponent CGF, which was also equipped with an on-board cannon. A CGF was destroyed if it was hit by five bullets. At each time step, Leuenberger and Wiering provided their agent with a reward proportional to the altitude achieved by the agent's CGF, with a maximum reward of 0.1 at the maximum altitude. This reward encouraged the agent to stay airborne despite the gravity and aerodynamics present in the environment. Additionally, Leuenberger and Wiering provided (a) a positive reward signal of $+5$ for each bullet that hit the opponent and (b) a negative reward signal of -25 if the agent was destroyed by the opponent.

(2a) Hindsight experience replay by Andrychowicz et al. (2017)

Andrychowicz et al. propose a method called hindsight experience replay. This method was specifically designed to overcome the problems posed by sparse binary rewards. The main idea behind hindsight experience replay is that every policy makes the learning agent successful at reaching some state, even if that state is not the desired state. Andrychowicz et al. speculated that by teaching the agent how to achieve different states, the agent will eventually learn to reach the desired state.

Hindsight experience replay was formulated as follows. The binary reward function $r(s_t, a_t)$ rewards the agent with a value of one if $s_t = g$, where g is some desired state. Otherwise, the reward is zero. The agent interacts with the environment during an episode (also called a trial or encounter), and receives $r_{t+1} = r(s_t, a_t)$ after every action it performs. Additionally, a sample of additional desired states G is drawn from the states that were observed by the agent. For each of these goal states $g' \in G$, the episode is replayed as though $g = g'$. Consequently, whenever the agent observes state s_t, it is rewarded with a value of one if $s_t = g'$. The hindsight experience replay method has been shown to outperform other methods of rewarding, e.g., performing a gripping task for a robot hand.

(2b) Backward learning by Edwards et al. (2018)

The method by Edwards et al. (2018) is based on the idea that it is possible to learn in two directions, namely forward and backward. Forward learning is the common way of doing reinforcement learning, viz. predicting what the next state will be and how much reward is obtained in that state, based on the current state and the possible actions. Backward learning involves observing a high-value state, and then theorising on which actions will most likely have led the agent to that state. Goyal et al. (2018) proposed a method similar to backward learning, and even in the same year.

Edwards et al. (2018) formulate backward learning as taking (a) a state s_{t+1} that has been observed, and (b) an action a_t that may have led to that state, and then predicting in which predecessor state $\hat{s}_t$ the action a_t was performed to arrive in s_{t+1}. By providing the agent with an additional reward $r(\hat{s}_t, a_t)$, the agent has an "imagined experience". Next, when the agent observes the real s_t, it will already have a good idea about which action to take to obtain the highest reward. To predict $\hat{s}_t$, Edwards et al. trained a neural network to model the function $b(s_{t+1}, a_t) \rightarrow \delta$, where δ is the difference between s_t and s_{t+1}. This way, the predicted predecessor state $\hat{s}_t$ can be calculated as $\hat{s}_t = s_{t+1} - b(s_{t+1}, a)$.

(2c) Non-uniform action-value initialisation by Sutton and Barto (2018)

Sutton and Barto warn against reward shaping because of the possible consequence of the agent getting stuck in a local optimum. They propose a simple alternative method, which is to initialise the action-value function (see Equations (4.2) and (4.4)) with an initial guess of the action-values. Traditionally, the action-values are initialised uniformly (e.g., all zero), so that the agent begins to explore the state space in an unbiased manner. By using non-uniform initial action-values, the policy of the agent is immediately guided into a particular direction. Since the advantage of this method is that the action-values are estimates, it holds that if the guess is wrong, the agent will correct the wrong estimates and continue as normal. However, making the guess requires an injection of human knowledge into the agent, which may be undesirable. Furthermore, for large and complex state spaces where the guesses are most needed, it is also the most difficult task to provide them.

4.3.4 Proposed solution: DOMAIN-REWARD

To counteract the problem of sparse rewards in our air combat simulations, we propose the following solution. We design a shaped reward function that provides rewards in encounters that are both won or lost. We call this reward function DOMAIN-REWARD. The main idea behind DOMAIN-REWARD is that the outcomes of encounters form a spectrum (see Figure 4.1). In this spectrum, the most undesirable behaviour is that the agent loses the encounter in a minimal amount of time, without having fired any missiles at the opponent. Similarly, the most desirable outcome is that the agent wins the encounter in a minimal amount of time, and requiring exactly one missile to destroy an opponent.

From the spectrum of outcomes, we derive three parameters for calculating the reward for an agent:

1. **The outcome of the encounter (r_{outcome}).** For simplicity, we model the parameter r_{outcome} as a binary reward signal. The parameter can either have a value of zero, if the agent lost the encounter, or a value of one, if the agent won the encounter.

2. **The duration of the encounter (r_{duration}).** We express r_{duration} as the ratio of the elapsed time to the total time available. In LWACS, the total time available for each encounter is 10 minutes of simulated time, after which the encounter ends automatically and counts as a loss for the red team. In case the agent lost the encounter, we prefer that this encounter was as long as possible, viz. r_{duration} approaches 1 (henceforth indicated by the $\rightarrow$ symbol). However, if the agent won the encounter, we prefer that the encounter was as short as

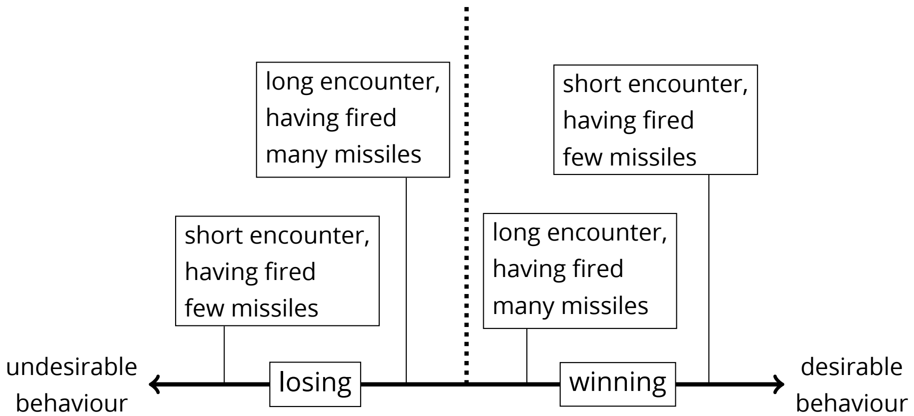

long encounter, having fired many missiles
short encounter, having fired few missiles
short encounter, having fired few missiles
long encounter, having fired many missiles
undesirable behaviour
losing
winning
desirable behaviour

Figure 4.1 The spectrum of desirable air combat behaviour.

possible, viz. $r_{\text{duration}} \rightarrow 0$. Therefore, if $r_{\text{outcome}} = 1$, we replace r_{duration} by $1 - r_{\text{duration}}$. This way, the agent is rewarded more for shorter encounters.

3. **The number of missiles that were fired (r_{missiles}).** We express r_{missiles} as the ratio of (a) the number of missiles that were fired by the agent's team to (b) the number of missiles that were available to that team at the start of the encounter (i.e., eight missiles for a two-ship). Similar to r_{duration}, we wish to differentiate between rewards for encounters that were lost or won. If the agent lost the encounter, we prefer if the agent's team had created firing opportunities for itself, and that the team made use of these opportunities by firing missiles. This would be indicated by a high ratio, resulting in $r_{\text{missiles}} \rightarrow 1$. In contrast, we claim that the best victories are the encounters in which the minimal number of missiles is needed to defeat the opponent. Therefore, if the agent won the encounter, we prefer $r_{\text{missiles}} \rightarrow 0$ and consequently substitute r_{missiles} by $1 - r_{\text{missiles}}$. However, in a two-versus-one scenario, at least one missile is necessary to defeat the opponent. To compensate for this necessary missile, we define an alternate form of r_{missiles}, which we call $r_{\text{missiles*}}$. If the agent's team has won, we calculate $r_{\text{missiles*}}$ as the ratio of (a) missiles that were fired by the agent's team to (b) the number of missiles that were available to that team at the start of the encounter minus one.

We introduce a weight factor for each parameter, so that the importance of each parameter can be controlled in the proposed reward function. For clarity, we show DOMAIN-REWARD as two functions: one that is used when the agent lost the encounter (see Equation (4.11)), and one that is used when the agent won the encounter (see Equation (4.12)).

$$r_{\text{terminal}} = w_1 * r_{\text{outcome}} + w_2 * r_{\text{duration}} + w_3 * r_{\text{missiles}} \tag{4.11}$$

$$r_{\text{terminal}} = w_1 * r_{\text{outcome}} + w_2 * (1 - r_{\text{duration}}) + w_3 * (1 - r_{\text{missiles*}}) \tag{4.12}$$

We formalise DOMAIN-REWARD and compare it to both BIN-REWARD and AA-REWARD in Section 4.5. Below, we briefly compare our proposed solution to the fitness function used by Yao et al. (2015). Although the functions are quite similar at first inspection, there are two important differences.

The first difference is that Yao et al. reward missiles that hit their target, but not the firing attempts. Furthermore, their function rewards agents for "winning more" (viz. with fewer missiles and more time spent safely) in a way that is similar to DOMAIN-REWARD, but no reward is provided if the agent loses the encounter. Our function rewards agents for the firing attempts regardless of the outcome of the missile. This way, we can reward agents that fire well, but still lose because of inherent random effects in the effectiveness of missiles. We further discuss these random effects in Section 4.4. By counting the firing attempts (see r_{missiles}), we are able to reward both (a) losing agents for making many attempts, one of which will eventually be successful over the course of many encounters, and (b) winning agents for needing as few missiles as possible to win.

The second difference is the use of the time parameter. In two-versus-one scenarios, the *time spent safely* parameter used by Yao et al. may provide a wrong indication of the success of the two-ship. It is possible that one agent places itself in temporary danger, so that the other agent can fire a successful shot at the opponent. This is why we consider only the entire duration of the encounter as an indicator of the success of the two-ship.

4.4 Unstable rewards

In this section we discuss the problem of unstable rewards. This section is outlined as follows. We begin by describing unstable rewards (Subsection 4.4.1). Next, we provide an overview of how unstable rewards are dealt with in the literature (Subsection 4.4.2). Finally, we propose our own solution to unstable rewards in the form of the AA-REWARD reward function (Subsection 4.4.3).

4.4.1 Problem description

In our air combat simulations, the environment is partially non-deterministic: whenever an agent observes s_t and performs a_t, it cannot be sure which s_{t+1} and r_{t+1} follow, even if the agent has previously performed a_t in s_t. This is somewhat confusing to the agent, as this means that in some cases the estimated value of a state can differ greatly from the reward that is actually obtained. In the literature, this is known as *probabilistic* or *unstable* rewards (cf. Tatsumi, Komine, Sato and Takadama, 2015; Chen, Yu, Da, Tan, Huang et al., 2018).

The cause of the unstable rewards in air combat simulations is the probabilistic behaviour of missiles. Air-to-air missiles are never one hundred percent reliable (see, e.g., Zheng and Feiguo, 2017). In order to be successful (viz. hit and destroy the intended target), each missile requires a

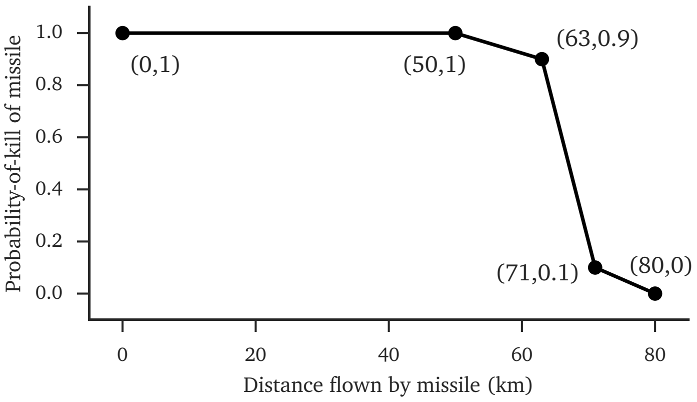


Figure 4.2 The probability-of-kill curve of the missiles in LWACS. The probability-of-kill of missiles depends on the number of kilometres flown by the missiles between (a) the moment of launch and (b) the moment of intercept. We define the curve by the following five points: (0,1), (50,1), (63,0.9), (71,0.1), and (80,0).

specific unbroken chain of events. This chain is called the *kill chain* (cf. Mills, 2009; MacLeod, 2012; Konokman, Kayran and Kaya, 2017). The probability of a particular missile completing an unbroken kill chain is called the *probability-of-kill* of that missile.

Depending on the application, the kill chain of a missile can include events with any level of granularity. For instance, MacLeod (2012) names three events: (a) the missile is loaded correctly and able to launch successfully, determined by the *probability-of-launch* P_L, (b) the missile reaches and hits its target, determined by the *probability-of-intercept* P_I, and (c) the target is destroyed upon a successful hit, determined by the *endgame probability-of-kill* P_K. The target of a missile is only destroyed if all three of these events occur.

LWACS uses a kill chain that differs slightly from the chain by MacLeod (2012). In LWACS it is assumed that all missiles launch successfully (i.e., $\mathrm{P}_L = 1$). Furthermore, the concepts of P_I and the endgame P_K are combined to form a "integrated" probability-of-kill. We define this probability-of-kill below.

Definition 4.4 (Probability-of-kill). The probability-of-kill (P_k) of a missile is the probability (in the range $[0, 1]$) that a missile destroys its target, given that the missile intercepts its target.

In LWACS, the P_k of a missile is calculated when the missile intercepts its target based on the distance the missile has flown to the target since its launch. The relationship between the distance flown and the P_k is shown in Figure 4.2. The maximum distance that can be flown by a missile is 80 km. After 80 km, we assume that the missile's fuel is depleted. Consequently, the missile is no longer able to intercept and destroy its target, and it is removed from the simulation.

The P_k of missiles cause unstable rewards in the following way. If the policy of an agent leads the agent towards firing a missile with a high P_k (i.e., the agent has a high probability of hitting the target, ending the encounter, and obtaining a high reward), there is still a chance that (a) the missile misses its target, and therefore that (b) the agent is not provided with the reward. Then, because executing this policy did not lead to a reward, the policy is adjusted to try other actions when the same states are encountered. However, the policy leading to this missile may have very well been a near-optimal policy. Conversely, missiles with a low P_k have a low probability of hitting their target, but may still lead to a reward. Therefore, the policy leading to the low P_k missile is reinforced and applied in the next encounter. However, it will now take the agent additional encounters to discover that this policy is actually a sub-optimal policy. In brief, the possibility of (a) unsuccessful high P_k missiles and (b) successful high P_k missiles frustrate the search for an optimal policy by requiring additional encounters to correctly estimate the success rate of missiles.

The unstable rewards that result from the probability-of-kill make firing missiles akin to the multi-armed bandit problem. In the multi-armed bandit problem (see, e.g., Kaufmann, Cappé and Garivier, 2016), a reinforcement learning agent has to repeatedly select one of n arms over a number of time steps. These arms function similarly to the arms on real-world slot machines. Upon selection of an arm, the arm provides a reward according to a hidden reward distribution. The goal of the agent is to maximise the total reward that is obtained (cf. Besbes, Gur and Zeevi, 2014; Joseph, Kearns, Morgenstern and Roth, 2016). The agent's problem is the choice between (a) spending time exploring the reward distributions of the arms, and (b) exploiting the arm that has so far been observed to provide the best rewards to obtain immediate rewards. In air combat simulations, the arms are the policies that lead to and include the action of firing a missile at a target. Theoretically, the agent has to follow each policy multiple times to accurately estimate the policy's value. However, in practice, the reinforcement learning algorithm changes the policy before the agent has a chance to do so.

4.4.2 Unstable rewards in the literature

Below, we provide an overview of three methods that have been presented in the literature with the goal of counteracting unstable rewards. These methods are (a) Double Q-learning by Van Hasselt (2010) and Van Hasselt (2011), (b) confidence intervals around rewards by Tatsumi et al. (2015), and (c) confidence intervals around state transitions by Tetreault, Bohus and Litman (2007).

(a) Double Q-learning by Van Hasselt (2010, 2011)

Van Hasselt proposes a variant of Q-learning called Double Q-learning. This variant addresses Q-learning's tendency to overestimate action values when unstable rewards are provided to the learning agent. The overestimated action values are the result of the argmax operator when actions are selected (see Equation (4.5)). Van Hasselt's solution is to maintain two Q-functions.

Double Q-learning works as follows. Two Q-functions are defined, called Q^A and Q^B. For each state s_t, an action a_t is selected by means of a combination of Q^A and Q^B, e.g., the average Q-value obtained from the two functions. This is shown in Equation (4.13).

$$a_t = \operatorname*{argmax}_{a_t} \frac{Q^A(s_t, a_t) + Q^B(s_t, a_t)}{2} \tag{4.13}$$

Then, after the action has been performed, either Q^A or Q^B is updated. Which of the two functions is updated can be chosen, e.g., in an alternating manner or at random. A difference with normal Q-learning is that the two Q-functions are updated using each other's Q-values. The update of Q^A is shown in Equation (4.14). The same is shown for Q^B in Equation (4.15).

$$\begin{gathered} Q^A(s_t, a_t) \leftarrow Q^A(s_t, a_t) + \eta\Big[r_{t+1} + \gamma Q^B(s_{t+1}, a^*) + Q^A(s_t, a_t)\Big] \\ \text{where } a^* = \operatorname*{argmax}_{a_{t+1}} Q^A(s_{t+1}, a_{t+1}) \end{gathered} \tag{4.14}$$

$$\begin{gathered} Q^B(s_t, a_t) \leftarrow Q^B(s_t, a_t) + \eta\Big[r_{t+1} + \gamma Q^A(s_{t+1}, b^*) + Q^B(s_t, a_t)\Big] \\ \text{where } b^* = \operatorname*{argmax}_{a_{t+1}} Q^A(s_{t+1}, a_{t+1}) \end{gathered} \tag{4.15}$$

Van Hasselt shows that the use of two Q-functions mitigates the bias that is caused by the use of the argmax operator. As a result, Double Q-learning converges faster on learning problems with unstable rewards than Q-learning does.

(b) Confidence intervals around rewards by Tatsumi et al. (2015)

Tatsumi et al. extended their learning classifier system (LCS, plural: LCSs) with the ability to form confidence intervals around expected rewards. Despite their name, LCSs are a family of rule-based evolutionary reinforcement learning algorithms. Tatsumi et al. used a variant called the accuracy-based learning classifier system (XCS). In an XCS, each rule is a state-action pair combined with a prediction p of the reward that is obtained by performing the action. The XCS keeps track of how accurate the predictions are. When the accuracy of a rule crosses a global lower limit ϵ_0, the rule is likely to be deleted and replaced by a newly generated rule. Tatsumi et al. found that unstable rewards caused low accuracy values in near-optimal rules, leading to the XCS impatiently deleting these rules. Therefore, the XCS never converged to a policy for some environments with unstable rewards.

To solve the problem of impatient deletions, Tatsumi et al. made two changes to the XCS. First, the global lower limit for the accuracy of each rule was replaced by a lower limit ϵ_0 per rule. Second, the lower limit of each rule was updated in an adaptive manner, so that each rule could independently learn to cope with unstable rewards. This was done by maintaining a function $S(s, a) \rightarrow \mathbb{R}$ that calculated the standard deviation of the rewards obtained by performing a in

s. This function was updated for each (s,a) pair until $S(s,a)$ converged for that pair. Then, the $S(s,a)$ was used to adjust the ϵ_0 of rules that fired on s to perform a, so that each rule became more "tolerant" of unstable rewards.

(c) Confidence intervals around state transitions by Tetreault et al. (2007)

Tetreault et al. proposed a method for the construction of confidence intervals over estimated cumulative rewards. The method was proposed in the context of a reinforcement agent learning the structure of spoken dialogue in cases where insufficient training data was available to generate reliable policies. The confidence intervals were constructed by counting the state transitions observed by the agent, e.g., by maintaining a function count(s_t, a_t, s_{t+1}). Using these counts, Tetreault et al. constructed a Dirichlet distribution. From this distribution, they sampled so-called transition tables that each provided a likelihood of transitioning between each s_t and s_{t+1}. The transition tables acted as possible models of the environment: by applying a policy π to each of the transition tables, an estimate of V^{π^*} was made while taking into account the possibility that the states in the environment transition in a different manner than the agent had observed so far.

4.4.3 Proposed solution: AA-REWARD

Our proposed solution is designing a reward function that rewards agents proportionally to the P_k of the missiles fired by the agents, rather than proportionally to the effect (i.e., the target is unharmed or destroyed) of the missiles. We call the reward function AA-REWARD, where AA stands for *air-to-air combat*. In AA-REWARD, we take into account the property that each missile has the ability to end the encounter. Therefore, it is preferable to end the encounter as soon as possible, by using as few missiles as possible.

We describe AA-REWARD's calculation of the reward on a high level. When the first missile intercepts its target, we assign a reward signal equal to the P_k of that missile to the shooter of the missile. Normally, when the first missile intercepts its target, the target would be destroyed and the encounter would end. However, as part of our proposed solution, we allow the encounter to continue with all participating CGFs as though the missile had failed to destroy its target. We continue to assign reward signals for subsequent missiles that intercept their targets. However, to acknowledge the ordering to the missiles and the ability of each missile to end the encounter (and thereby ending the gathering of reward in this encounter), we reduce the reward that can be earned after each intercept.

The reward that can be earned is reduced as follows. We define a maximally obtainable reward $m_1 = 1$. When the first missile intercepts its target, a reward signal is assigned that is equal to $r_1 = \mathrm{P}_{k1} * m_1$. Here, P_{k1} is the P_k of the first missile. Next, the maximally obtainable reward is reduced to $m_2 = m_1 - r_1$. For each subsequent missile, the reward signal is calculated in a similar manner: $r_i = \mathrm{P}_{ki} * m_i$, after which the maximally obtainable reward is recursively reduced to $m_{i+1} = m_i - r_i$.

Table 4.1 A comparison of BIN-REWARD, DOMAIN-REWARD, and AA-REWARD.

	BIN-REWARD	DOMAIN-REWARD	AA-REWARD
Description	Rewards winning an encounter	Rewards (a) winning an encounter, (b) use of time, and (c) use of missiles	Rewards proportionally to P_k values of missiles
Reward sparsity	Sparse	Least sparse	Somewhat sparse
Reward stability	Unstable	Somewhat unstable	Stable
On intercept of missile	The encounter terminates	The encounter terminates	The encounter continues

For instance, if the first missile that intercepts its target only does so with a low $\mathrm{P}_{k1} = 0.1$, a reward signal $r_1 = 0.1 * m_1 = 0.1$ is assigned to the shooter of the missile. This missile only had a small chance of ending the encounter. For the next missile, the maximally obtainable reward remains large: $m_2 = m_1 - 0.1 = 0.9$. If the second missile intercepts its target with a high $\mathrm{P}_{k2} = 0.8$, the shooter of the second missile is now assigned a reward of $0.8 * 0.9 = 0.72$, and m_3 becomes $0.9 - 0.72 = 0.18$.

Compared to the binary reward function, we introduce a gradient into the rewards. The gradient gives the agent more fine-grained feedback on the effectiveness of the behaviour leading up to the P_k of each missile. By letting missiles leave their targets unharmed upon intercept, agents are able to gather more information about the effectiveness of their behaviour. We believe that our proposed solution is an elegant manner to (a) remove subjectively weighted factors such as in shaped reward functions, and (b) remove the concept of winning and losing from the learning process. Thereby, the agents can focus on learning to fire effective missiles, while receiving informative rewards by which the agents can improve their behaviour. We formalise AA-REWARD and compare it to both BIN-REWARD and DOMAIN-REWARD in Section 4.5.

4.5 Overview of the three reward functions

In this section, we provide an overview of the three reward functions named in this chapter: (1) BIN-REWARD, the binary reward function, (2) DOMAIN-REWARD, our solution to sparse rewards, and (3) AA-REWARD, our solution to unstable rewards. A comparison of the three reward functions is given in Table 4.1. Below, we describe each of the three reward functions separately. We begin by briefly restating BIN-REWARD (Subsection 4.5.1). Next, we provide formal descriptions of DOMAIN-REWARD (Subsection 4.5.2) and AA-REWARD (Subsection 4.5.3).

4.5.1 BIN-REWARD

The binary reward function BIN-REWARD provides rewards for winning encounters. BIN-REWARD is shown in Equation (4.16).

$$\text{BIN-REWARD}(s_{\text{terminal}}) = \begin{cases} 1 & \text{if the agent won the encounter} \\ 0 & \text{otherwise} \end{cases} \tag{4.16}$$

The rewards provided by BIN-REWARD are sparse: the rewards are only provided (a) in the terminal states, and only if (b) the encounter was won by the agent's team. Furthermore, the rewards provided by BIN-REWARD are unstable: the same policy that causes the team to win one encounter, may cause the same team to lose the next encounter. This instability is caused by the P_k of missiles, as discussed in Section 4.4.

4.5.2 DOMAIN-REWARD

The reward function DOMAIN-REWARD provides rewards for (a) winning encounters, (b) efficient use of missiles, and (c) efficient use of time.

We define efficient use of missiles as follows: if an encounter was won, missiles were used most efficiently if only one missile was required to destroy the opponent. Conversely, if an encounter is lost, missiles were used most effectively if as many missiles were fired as possible. When an agent has fired all of its missiles, the agent has created many firing opportunities for itself, each of which with the potential to win the encounter.

Moreover, we define efficient use of time as follows: if an encounter was won, time was used most efficiently if the encounter was won as fast as possible. Conversely, if an encounter is lost, time was used most efficiently if the encounter was lost after staying alive as long as possible. When an agent stays alive as long as possible, the agent likely has (a) effective defensive behaviour, as well as (b) the ability to create the most firing opportunities for itself.

For clarity, we split DOMAIN-REWARD into two functions Equation (4.17). The functions are called DOMAIN-REWARD$^-$ and DOMAIN-REWARD$^+$. Equation (4.18) and Equation (4.19) formalise these two functions, respectively.

$$\text{DOMAIN-REWARD}(s_{\text{terminal}}) = \begin{cases} \text{DOMAIN-REWARD}^-(s_{\text{terminal}}) & \text{if the agent lost} \\ \text{DOMAIN-REWARD}^+(s_{\text{terminal}}) & \text{if the agent won} \end{cases} \tag{4.17}$$

$$\text{DOMAIN-REWARD}^-(s_{\text{terminal}}) = \frac{1}{8}\frac{m_{\text{fired}}}{m_{\text{starting}}} + \frac{1}{8}\frac{t_{\text{duration}}}{t_{\max}} \tag{4.18}$$

$$\text{DOMAIN-REWARD}^+(s_{\text{terminal}}) = \frac{3}{4} + \frac{1}{8}\max(0, 1 - \frac{m_{\text{fired}}}{m_{\text{starting}-1}}) + \frac{1}{8}\Big(1 - \frac{t_{\text{duration}}}{t_{\max}}\Big) \tag{4.19}$$

In Equation (4.18) and Equation (4.19), (a) m_{fired} is the number of missiles fired by the agent's team, (b) m_{starting} is the number of missiles that was available to the agent's team at the start of the encounter, (c) $m_{\text{starting}} - 1$ is m_{starting} minus one, (d) t_{duration} is the duration of the encounter, and (e) t_{max} is the maximal duration of the encounter.

The rewards provided by DOMAIN-REWARD are less sparse than those provided by BIN-REWARD. The rewards are still only provided in the terminal state. However, DOMAIN-REWARD provides rewards for both winning and losing. Compared to BIN-REWARD, we no longer provide a uniform reward signal for winning. Instead, the reward for winning an encounter is defined by a spectrum that is characterised by the agents' use of missiles and time (see Equation (4.19)). This spectrum ranges from "barely winning" (viz. a low reward for winning after firing many missiles, and late in the encounter) to "winning convincingly" (viz. a high reward for winning after firing few missiles, and early in the encounter). A similar spectrum is used for providing a reward for losing, ranging from "losing decisively" (viz. a low reward for losing after firing few missiles, and early in the encounter) to "barely losing" (viz. a high reward for winning after firing many missiles, and late in the encounter).

Since the rewards provided by DOMAIN-REWARD still depend on whether the agents won or lost the encounter, the rewards are unstable. However, compared to BIN-REWARD, the effect of the unstable rewards is somewhat mitigated by the use of shaped rewards.

4.5.3 AA-REWARD

The reward function AA-REWARD provides rewards to agents proportionally to the P_k values of the missiles fired by the teams of the agents. We formalise AA-REWARD as follows. Whenever a missile successfully intercepts its target, we assign a reward signal $r_{\text{intercept}}$ to the shooter of the missile. The assigned reward signals are not immediately provided to the agents. Instead, the reward signals are provided when a terminal state is reached.

The calculation of $r_{\text{intercept}}$ for the nth missile intercept is shown in Equation (4.20). Each encounter, we define a maximally available reward signal of 1. This is the total reward signal that is available in the encounter. Because of the recursive summation used in Equation (4.20), the $r_{\text{intercept}}$ for each missile intercept depends on all missile intercepts that came before it during the encounter. For the first missile, $r_{\text{intercept}}(1) = \mathrm{P}_{k1}$, as $\sum_{m\geq 1}^{0} r_{\text{intercept}}(m) = 0$ by definition. This missile has the first opportunity to end the encounter, and therefore has the potential to secure the largest part of the maximally available reward signal as a reward signal for the shooter. With each missile intercept, the maximally available reward signal decreases, reflecting the importance of firing missiles with high P_k values before the opponent does.

$$r_{\text{intercept}}(n) = \mathrm{P}_{kn}(1 - \sum_{m=n-1}^{0} r_{\text{intercept}}(m)) \tag{4.20}$$

Since team behaviour is an important concept in air combat (see Chapter 3), we sum the reward signals that are assigned to each member of a team, and then provide to each member the summed reward signal. The summation is shown by Equation (4.21). Here, team(n) holds if the nth missile was fired by the team of the agent that is being rewarded. Thus, agents are rewarded for both (a) firing missile with a high P_k and (b) enabling team members to do so.

$$\textsc{aa-reward}(s_{\text{terminal}}) = \sum_{\{n \mid \text{team}(n)\}} r_{\text{intercept}}(n) \tag{4.21}$$

Regarding the sparsity of the rewards, the rewards provided by AA-REWARD are by definition less sparse than the rewards provided by BIN-REWARD. Whereas BIN-REWARD only rewards missile that destroy their target, AA-REWARD rewards each missile that intercepts its target, regardless of whether the target is destroyed upon intercept. However, the rewards provided by AA-REWARD remain more sparse than those provided by DOMAIN-REWARD. A team that fires no missiles during an encounter, and therefore loses the encounter, will be provided a reward signal of zero by AA-REWARD. In contrast, DOMAIN-REWARD will still provide the team with some reward signal for the duration of the encounter.

Furthermore, we observe that out of the three reward functions, the rewards provided by AA-REWARD are the most stable. In AA-REWARD, the reward signals no longer depend on winning and losing encounters. Instead, we have turned the probabilities of winning and losing into the reward signals that are provided to the agents. This way, a missile fired in some state s by agent a will always result in the same reward signal for a. The reward signals stimulate the agents to fire missiles with high P_k values. By firing missiles with high P_k values, the agents automatically increases their chances of destroying their opponents and winning encounters.

4.6 Experimental setup

To determine to what extent the use of DOMAIN-REWARD and AA-REWARD lead to improved performance over the use of the simple BIN-REWARD function we designed an experiment. The experiment consists of automated simulations. In this section, we present the setup of the experiment. The setup is largely similar to the setup presented in Chapter 3. It is divided into five parts: a description of the red team (Subsection 4.6.1), the blue team (Subsection 4.6.2), the scenarios that were used (Subsection 4.6.3), the independent and dependent variables (Subsection 4.6.4), and a description of our method of analysis (Subsection 4.6.5).

4.6.1 Red team

The *red team* (henceforth: *red*) consists of two fighter jet CGFS, a lead and a wingman. The capabilities of the CGFS are described in Appendix A.2. The goal of the red team is to learn how to defeat the blue team in three different scenarios (see Appendix A.4). The red team uses the

DECENT coordination method (see Chapter 3) and learns by means of one of the three reward functions presented in this chapter: (1) BIN-REWARD, (2) DOMAIN-REWARD, and (3) AA-REWARD.

4.6.2 Blue team

The *blue team* (henceforth: *blue*) consists of a single fighter jet CGF. The capabilities of the CGF are described in Appendix A.2. The goal of the blue team is to defeat the red team, by hitting one of the reds with a missile. The behaviour of the blue CGF is governed by scripts (see Appendix A.3).

4.6.3 Scenarios

In the automated simulations that are performed, we use the four two-versus-one scenarios described in Appendix A.4: (1) the basic scenario, (2) the close range scenario, (3) the evasive scenario, and (4) the mixed scenario. These four scenarios are repeatedly presented to red. This way, red is able learn how to behave in each of the four scenarios over a run of encounters.

4.6.4 Independent and dependent variables

The experiment uses two independent variables: (1) the three reward functions (BIN-REWARD, DOMAIN-REWARD, and AA-REWARD), and (2) the four scenarios (basic, close range, evasive, and mixed). The combination of these independent variables results in a 3×4 fully factorial design with twelve conditions. For each condition, we register the win rates of red. The win rates are the dependent variable in the experiment.

4.6.5 Method of analysis

We make use of two measures to analyse the win rates of red: we calculate (1) the final performance, and (2) the turning points. The two measures are explained in Subsection 3.3.6. As in Chapter 3, we investigate the results from the two measures by means of an ANOVA. Using the ANOVA, we determine whether red's final performance and/or turning points differ between the three reward functions and/or four scenarios.

4.7 Results

In this section, we present the results of the experiment. For each condition, we simulated 150 runs of encounters. The runs initially consisted of 100 encounters in each scenario. However, because it seemed that the performance of the CGFs in (1) the evasive scenario and (2) the mixed scenario had not yet levelled off after 100 encounters, we extended these runs by 50 encounters. In total, we simulated 75,000 encounters for the experiment.

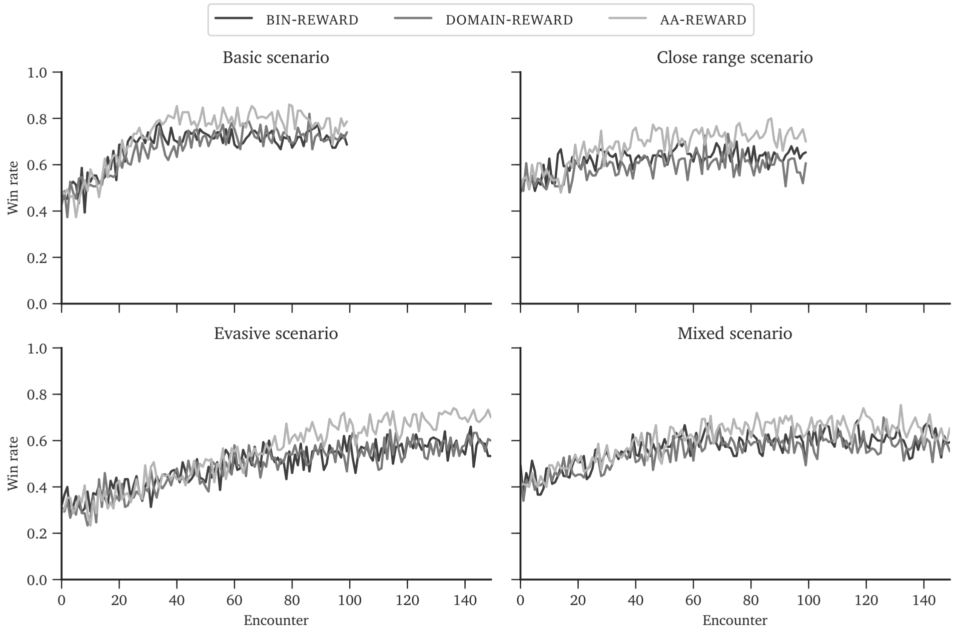


Figure 4.3 The win rates of red against blue. Red used one of three reward functions: (1) BIN-REWARD, (2) DOMAIN-REWARD, and (3) AA-REWARD. The behaviour of blue depended on four scenarios: (1) the basic scenario, (2) the close range scenario, (3) the evasive scenario, and (4) the mixed scenario.

Figure 4.3 shows the win rates of red against blue. The win rates are divided over the four scenarios. For each scenario, the win rates of red using each of the three reward functions are shown. The final performance and the turning points of red are shown in Table 4.2 and Table 4.3, respectively. Two two-way ANOVAs were performed.

Table 4.2 The final performance of red. A higher final performance is better.

	Basic scenario		Close range scenario		Evasive scenario		Mixed scenario		Grand mean	
Reward function	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
BIN-REWARD	.715	.135	.645	.132	.571	.165	.609	.120	.635	.148
DOMAIN-REWARD	.728	.151	.610	.139	.574	.150	.586	.119	.625	.153
AA-REWARD	.792	.113	.723	.112	.683	.133	.660	.107	.715	.127

Table 4.3 The turning points of red. A lower turning point is better.

	Basic scenario		Close range scenario		Evasive scenario		Mixed scenario		Grand mean	
Reward function	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
BIN-REWARD	19.7	10.8	16.5	8.0	48.0	32.4	27.0	17.3	27.8	23.1
DOMAIN-REWARD	20.8	13.3	18.6	9.5	51.0	33.4	31.1	25.8	30.4	26.0
AA-REWARD	19.0	8.3	15.9	7.2	47.2	27.7	25.7	15.1	26.9	20.7

First two-way ANOVA (final performance)

The first two-way ANOVA was conducted on the influence of the two independent variables (reward function, scenario) on the final performance of red. All effects were statistically significant at the $\alpha = .05$ level. The main effect of reward function yielded an F ratio of $F(2, 1788) = 82.851$, $p < .001$, indicating a significant difference between the final performances of red using each of the three reward functions. The main effect of scenario yielded an F ratio of $F(3, 1788) = 98.510$, $p < .001$, indicating a significant difference in the final performance of red in each of the four scenarios. Furthermore, the interaction between the reward functions and the scenarios was significant, $F(6, 1788) = 2.490$, $p = .021$. We performed a post hoc Tukey HSD test (see, e.g., Holmes et al., 2016) to determine the significant differences in final performance between specific pairs of (a) reward functions, and (b) scenarios. First, the post hoc test revealed that (a) the final performance differed significantly between AA-REWARD and the other two reward functions. Second, the post hoc test revealed that (b) the final performance differed significantly between all scenarios, $p < .001$, except between the evasive and mixed scenarios.

Second two-way ANOVA (turning points)

The second two-way ANOVA was conducted on the influence of the two independent variables (reward function, scenario) on the turning points. All effects were statistically significant at the $\alpha = .05$ level. The main effect of reward function yielded an F ratio of $F(2, 1788) = 4.901$, $p = .008$, indicating a significant difference between the turning points of red using each of the three reward functions. The main effect of scenario yielded an F ratio of $F(3, 1788) = 236.285$, $p < .001$, indicating a significant difference between the turning points of red in the four scenarios. We performed a post hoc Tukey HSD test to determine the significant differences in the turning points between specific pairs of (a) reward functions, and (b) scenarios. First, the post hoc test revealed that (a) the turning points differed significantly between AA-REWARD and DOMAIN-REWARD, $p = .008$. Consequently, the turning points did not differ significantly between AA-REWARD and BIN-REWARD. Second, the post hoc test revealed that (b) the turning points differed significantly between all scenarios, except between the basic scenario and the close range scenario.

4.8 Discussion

In this section, we discuss the results of our experiment. Specifically, we cover three topics. First, we discuss the results achieved by use of DOMAIN-REWARD (Subsection 4.8.1) and AA-REWARD (Subsection 4.8.2). We conclude the discussion by reviewing the sparsity and stability of the rewards offered by the reward functions (Subsection 4.8.3).

4.8.1 Using DOMAIN-REWARD

Both DOMAIN-REWARD and AA-REWARD were intended to improve upon BIN-REWARD, by improving the sparseness and the stability of the rewards, respectively. However, DOMAIN-REWARD does not deliver improved results: the final performance achieved by CGFS is on average lower than the final performance achieved by CGFS that make use of BIN-REWARD, albeit not by a statistically significant amount. Still, a statistically significant difference is found in the turning points, which in the case of DOMAIN-REWARD are worse than the turning points achieved by using either BIN-REWARD or AA-REWARD. In other words, DOMAIN-REWARD does not improve upon BIN-REWARD. Of each reward value produced by DOMAIN-REWARD, a part equating 75% of the value is calculated in the same manner as BIN-REWARD calculates its reward value (see Equation (4.18) and Equation (4.19)). Therefore, it appears that the worse performance of DOMAIN-REWARD is caused by the two extra factors we incorporated into the reward value: (a) the use of missiles, and (b) the use of time. These findings once again yet inadvertently highlight the difficulty of designing reward functions.

4.8.2 Using AA-REWARD

Overall, the use of AA-REWARD results in the highest performance gain. The increase in performance over BIN-REWARD is 12.6%. Interestingly, Figure 4.3 shows that the final performance achieved by use of AA-REWARD somewhat matches that of BIN-REWARD and DOMAIN-REWARD during the first 40-80 encounters in each of the four scenarios. In the encounters that follow, the final performance diverges from that of the other two reward functions. This may indicate that the stability of the rewards offered by AA-REWARD only pays off in terms of performance after some time has passed. It is possible that until then, the relative sparsity of these rewards keeps the behaviour of the CGFS from quickly moving in an optimal direction. Instead, the improvements in behaviour appear to be steady and cumulative, such as can be seen in the clear upward trend in performance in the evasive scenario (see Figure 4.3). Despite the slow divergence of the performance, the results show that overall, CGFS using AA-REWARD do not learn any slower than CGFS using BIN-REWARD. Table 4.3 and the second ANOVA indicate that the number of encounters needed by these CGFS to reach their turning points does not differ significantly.

4.8.3 Sparsity and stability

As shown in Table 4.1, BIN-REWARD provides the most sparse and most unstable reward to CGFS. Of the two functions that we newly designed in this chapter, only one lead to a significant performance increase over BIN-REWARD, namely AA-REWARD. The less sparse but still quite unstable rewards provided by DOMAIN-REWARD resulted in no increase in final performance and worse turning points than BIN-REWARD. The somewhat sparse but stable rewards provided by AA-REWARD resulted in an increase in final performance, but no improvement in the turning points. Broadly speaking, it looks like (a) dense rewards lead to worse turning points, and (b) stable rewards lead to the highest final performance. However, our sample size of different reward functions is too small to draw accurate conclusions regarding the specific effects of dense and stable rewards. Furthermore, because DOMAIN-REWARD was manually designed, there may exist variations of the weights and factors that make up the rewards (see Subsection 4.5.2) that maintain its reward density but lead to better performance.

4.9 Answering research question 2

In this chapter, we investigated the effects of reward functions on the performance of air combat CGFS. Specifically, we addressed research question 2: *To what extent can we improve the reward function for air combat CGFS?* The research question refers to BIN-REWARD, a binary reward function that offers sparse and unstable rewards to the CGFS. To answer research question 2, we developed two new reward functions. The first new reward function, DOMAIN-REWARD, offers the least sparse rewards, but the rewards remain relatively unstable. The second new reward function, AA-REWARD, offers rewards that are less sparse than those offered by BIN-REWARD, and the rewards are completely stable. We used both new reward functions in automated air combat simulations, and compared the performance achieved by the CGFs with the performance achieved by using BIN-REWARD. While DOMAIN-REWARD fails to significantly improve the performance of the CGFs, the use of AA-REWARD leads to a 12.6% increase in final performance. Therefore, we answer research question 2 as follows: by replacing a simple binary reward function (i.e., BIN-REWARD) by a reward function that is tailored to the air combat domain (i.e., AA-REWARD), we are able to improve the effectivess of the generated behaviour of air combat CGFs by 12.6%.

5 Transfer of knowledge between scenarios

In this chapter we investigate research question 3: *To what extent can knowledge built with dynamic scripting be transferred successfully between CGFs in different scenarios?*

As the complexity of air combat scenarios increases, so does the time to learn good behaviour in these scenarios. Decreasing the learning time in complex scenarios may be possible by reusing the knowledge about air combat that has been stored in previously generated behaviour models. The reuse of previously gained knowledge in order to learn and solve new problems is known as *transfer learning* (see Definition 5.1). Transfer learning has been shown to have the potential of shortening the learning time between domains that are sufficiently similar (cf. Lu, Behbood, Hao, Zuo, Xue et al., 2015). In this chapter, we examine a particular use case, namely the transfer of knowledge from air combat CGFs that have learned to win two-versus-*one* scenarios, to CGFs that have to learn how to win two-versus-*two* scenarios. Next, we compare the performance of the CGFs with the transferred knowledge to that of CGFs that learn to behave from scratch in the two-versus-two scenarios. By doing so, we aim to determine to what extent the previously built knowledge aids in the development of behaviour models in the two-versus-two scenarios.

This chapter is organised as follows. First, we introduce the concept of transfer learning (Section 5.1). Next, we present the use case that we use as the foundation of our study of transfer learning in this chapter (Section 5.2). Guided by the use case, we conduct an experiment in which we transfer knowledge between two distinct two-ships of CGFs and then determine the success of the transfer (Section 5.3). We present the results of the experiment (Section 5.4) and discuss them (Section 5.5). Finally, we summarise the chapter and answer research question 3 (Section 5.6).

This chapter is based on the following publication.

- A. Toubman, J. J. Roessingh, P. Spronck, A. Plaat and H. J. Van den Herik (2015b). Transfer Learning of Air Combat Behavior. In: *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. Miami, Florida: IEEE Press, pp. 226–231. DOI: 10.1109/ICMLA.2015.61

5.1 The concept of transfer learning

Transfer learning is a range of methods for reusing knowledge and skills that were gained when performing one task, and that are now applied on another task (cf. Taylor and Stone, 2009; Pan and Yang, 2010; Lu et al., 2015). The idea behind transfer learning is that it is easier for an agent (e.g., a CGF, robot, or software program) to perform a new task when the agent has already learned to perform a similar task. We refer to the task which the agent has already learned to perform as the *source task*. The new task is referred to as the *target task*. Below, we define the concepts of (1) transfer learning, (2) source task, and (3) target task.

Definition 5.1 (Transfer learning). Transfer learning is defined as learning to perform a target task by reusing knowledge that was previously gained on a source task.

Definition 5.2 (Source task). In the context of transfer learning, a source task is an intermediate task that a CGF has to learn to perform well, in order for the CGF to gain knowledge that may be useful for performing a different (but still similar) task.

Definition 5.3 (Target task). In the context of transfer learning, the target task is the task of interest, viz. the task that has similarities with the source task and that we desire to be performed by a CGF.

In the remainder of this section, we continue our introduction of transfer learning by briefly discussing three related topics: transfer learning methods (Subsection 5.1.1), transfer learning in reinforcement learning (Subsection 5.1.2), and transfer learning in dynamic scripting (Subsection 5.1.3). Finally, we conclude the section with a note on the burden of human knowledge (Subsection 5.1.4).

5.1.1 Transfer learning methods

Transfer learning methods have been successfully applied in classification, regression and clustering tasks (cf. Lu et al., 2015; Shao, Zhu and Li, 2015; Day and Khoshgoftaar, 2017). In these tasks, due to model availability and the time it takes to train new models, it can be desirable to reuse old models on new data. However, when the new data has different features or a different distribution, the models will have to be adapted. In these cases, the knowledge stored in the old models should be reused as efficiently as possible. The research on transfer learning methods concerns itself with studying effective ways for the reuse of this knowledge.

An example of transfer learning in practice is the work by Ferrucci, Brown, Chu-Carroll, Fan, Gondek et al. (2010). In 2011, an artificial intelligence system called Watson competed with human contestants in the open-domain question-answering television program *Jeopardy!* (Ferrucci et al., 2010). As part of Watson's preparation, Ferrucci et al. tested Watson's question-answering capabilities on various question-answer databases. In one instance, Watson's capabilities improved significantly on a new, closed-domain database, by first allowing it to learn to answer *Jeopardy!*-style questions.

5.1.2 Transfer learning in reinforcement learning

Transfer learning has been identified as a useful tool in reinforcement learning (see, e.g., Lazaric, 2012; Bianchi, Celiberto Jr, Santos, Matsuura and De Mantaras, 2015; Hou, Ong, Feng and Zurada, 2017a; Spector and Belongie, 2018). In reinforcement learning, transfer learning enables the reuse of previously learned behaviours (Bou Ammar, Chen, Tuyls and Weiss, 2014). For example, we consider an air combat scenario as the source task, and a more difficult air combat scenario as the target task. In both tasks, the learning CGF needs to discover that it should (1) fire missiles in order to win the scenario, and (2) evade missiles that were fired by the opponent(s) in order not to lose the scenario.

If the source task is relatively easy to complete, we may expect that the learning CGF will quickly discover relevant behaviours for which the CGF will be rewarded. When the CGF (including its knowledge) is transferred from the source task to the target task, the CGF may find that the behaviours for which it was rewarded in the source task are also applicable in the target task. Applying that knowledge then may speed up learning the remaining behaviours that are needed to perform the target task. If the CGF is not transferred, it starts with a *tabula rasa* instead, and then needs to begin to discover the behaviours that are necessary to win the scenario.

5.1.3 Transfer learning in dynamic scripting

The dynamic scripting algorithm, as we have used it in our research thus far, requires predefined knowledge (in the form of rules in a rulebase) to function. While learning to perform a task, a CGF using dynamic scripting builds up new knowledge about which rules are required to perform the task. This knowledge is stored as the weights that are associated to the rules. In the context of air combat simulations, the tasks (viz. the scenarios) that we use in our experiments are quite similar. For instance, in all scenarios, the opponent has to be hit by a missile, before the learning CGF is hit by one of the opponent's missiles. Therefore, rules that, e.g., enable missile-firing behaviour are required to have high weights in order to win each scenario.

An interesting question that now arises is the following: in the air combat domain, to what extent does the knowledge that was built up in one scenario (i.e., the source task) affect the performance and any further learning in a second scenario (i.e., the target task)? On the one hand, if a transfer of knowledge in this domain leads to higher performance on the target task at a faster rate than CGFs that have to learn to perform the target task starting with zero knowledge, such a transfer may speed up the development of challenging CGFs for real-world training simulations. On the other hand, if we find that the knowledge brought along from an earlier air combat scenario hampers the performance on the next scenario (see Subsection 5.1.4), measures should be taken to erase this knowledge between scenarios.

5.1.4 The burden of human knowledge

In contrast to the expectation that learning to perform the source task aids in learning to perform a related target task, the literature has also shown that it is possible that too much (human) knowledge becomes a burden for an agent such as a CGF. A high-profile example of the "burden of knowledge" is the difference between the ALPHAGO program (Silver et al., 2016), and the related ALPHAGO ZERO program (Silver et al., 2017b). Both ALPHAGO and ALPHAGO ZERO are machine learning programs that are designed to play the game of Go by means of a combination of deep neural networks and Monte-Carlo tree search. The ALPHAGO program learned successfully to play Go by using some 24 million recorded games that were played by humans as training data, and then using some 16 million games from self-play. This was sufficient to defeat the reigning world Go champion Ke Jie (see, e.g., Chao, Kou, Li and Peng, 2018). However, a superior ALPHAGO ZERO was developed thereafter. The new program learned to play Go by self-play only, i.e., by starting *tabula rasa* and having only the rules of the game at its disposal.

Furthermore, a more generalised version of ALPHAGO ZERO called ALPHAZERO was able to play two games in addition to Go, namely the games of chess and shogi (Silver, Hubert, Schrittwieser, Antonoglou, Lai et al., 2017a). This was achieved in part by taking away assumptions about the game of Go, such as symmetry of the board caused by certain reflections and rotations. In summary, the ALPHAGO family of programs is an example of how injected knowledge and biases can hamper performance. It remains to be seen whether this is also the case in the learning of air combat behaviour. In any case, we should be aware of the fundamental difference between human knowledge and machine-generated knowledge from scratch.

5.2 Use case

In this section, we present our use case for transfer learning in air combat simulations. Below, we first describe the use case (Subsection 5.2.1). Then, we explain our implementation of the use case using dynamic scripting (Subsection 5.2.2).

5.2.1 Description

The use case entails the transfer of knowledge built up by a two-ship of CGFs to a second two-ship. The first two-ship (henceforth: the reds′) builds up its knowledge by learning to defeat an opponent in a two-versus-one air combat scenario. The second two-ship (henceforth: the reds″) uses this knowledge to learn to defeat *two* opponents in four different two-versus-two scenarios. Thus, in our use case the two-versus-one scenario is the source task. By extension, the two-versus-two scenarios are the target tasks.

In order to determine to what extent the transferred knowledge benefits the performance of the reds″ in the two-versus-two scenarios, we introduce a third two-ship (henceforth: the

reds_0). Like the reds$''$, the reds_0 learn to defeat two opponents in the two-versus-two scenarios. However, unlike the reds$''$, the reds_0 do so without any transferred knowledge (hence the zero in the name).

5.2.2 Implementation in dynamic scripting

In this subsection, we describe how we implement the use case by means of CGFs that learn by dynamic scripting. In brief, we implement a transfer of knowledge between two CGFs that learn by means of dynamic scripting by copying the rulebase, including the weights, from one CGF to the other CGF. The complete implementation of the use case consists of three steps. Below, we describe each of the three steps.

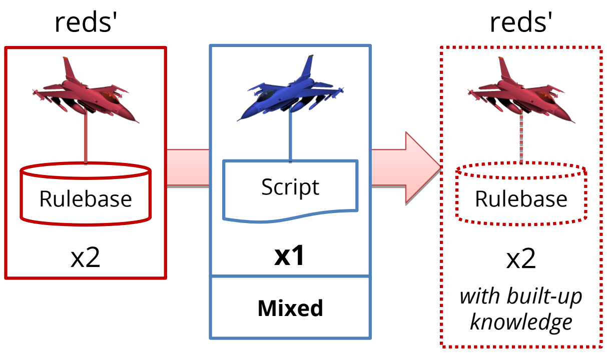


Figure 5.1 Step 1 of the implementation of the use case. A two-ship of red CGFs (the reds$'$) learns to defeat a blue CGF in the two-versus-one mixed scenario. The learning process of the reds$'$ leads to new knowledge in the rulebases (in the form of weights).

Step 1. The reds$'$ build up knowledge in a two-versus-one scenario. The reds$'$ learn to defeat a blue CGF in the mixed two-versus-one scenario. The mixed scenario is combination of three two-versus-one scenarios: (1) the basic scenario, (2) the close range scenario, and (3) the evasive scenario. The mixed scenario and its constituent scenarios are described in Appendix A.4.1. The reds$'$ learn which rules are useful for defeating the opponent. This knowledge is stored in the form of the weights that are attached to the rules in the rulebases of the reds$'$. Figure 5.1 shows Step 1 graphically.

Step 2. The reds$''$ use transferred knowledge in two-versus-two scenarios. The rulebases of the reds$'$ (see Step 1) are copied to the reds$''$. The reds$''$ use the copied rulebases as their initial knowledge for learning to defeat two blue CGFs in four distinct two-versus-two scenarios. The two-versus-two scenarios are described in Appendix A.4.2. Three of the scenarios are based on the two-versus-one scenarios that the reds encountered as part of

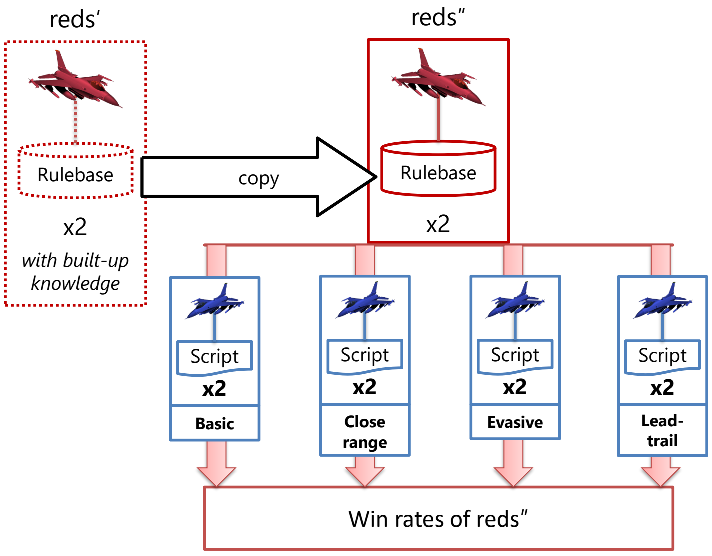


Figure 5.2 Step 2 of the implementation of the use case. The rulebase of the reds′ (resulting from Step 1) is copied to the reds″. The reds″ use this rulebase to learn to defeat two opponents, in each of four two-versus-two scenarios: the basic scenario, the close range scenario, the evasive scenario, and the lead-trail scenario. We record and store the win rates of the reds″ in each of the four scenarios.

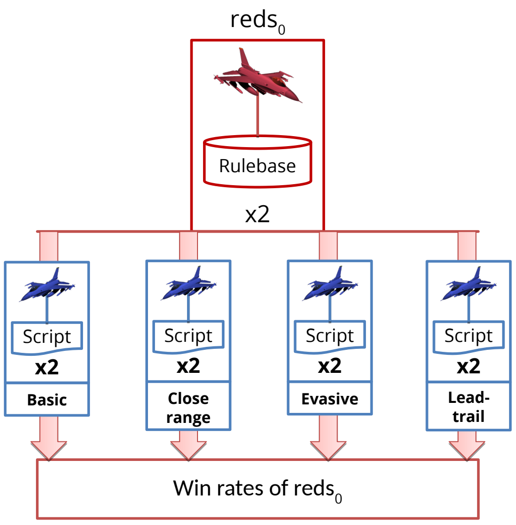


Figure 5.3 Step 3 of the implementation of the use case. The reds$_0$ learn to defeat two opponents, in each of four two-versus-two scenarios: the basic scenario, the close range scenario, the evasive scenario, and the lead-trail scenario. In contrast to Step 2, the knowledge of the reds′ is *not* transferred to the reds$_0$. We record and store the win rates of the reds$_0$ in each of the four scenarios.

the mixed scenario: (a) the basic scenario, (b) the close range scenario, and (c) the evasive scenario. The fourth scenario is (d) the lead-trail scenario. This is a new scenario, in which the blue lead approaches the reds$''$, while the blue wingman follows closely behind. In each of the four scenarios, we record and store the win rates of the reds$''$, i.e., how often the reds$''$ defeat their opponents throughout the learning process (see Subsection 3.3.5). Figure 5.2 shows Step 2 graphically.

Step 3. The reds$_0$ perform the two-versus-two scenarios without transferred knowledge. Step 3 is similar to Step 2. A two-ship of reds, in this case the reds$_0$, are placed in four two-versus-two scenarios. The reds$_0$ have to learn to defeat the two opponents in each of the scenarios. However, the reds$_0$ have to do so from scratch, viz. with a newly initialised rulebase that does not contain any previously built-up knowledge. We collect the win rates of the reds$_0$ so that they may be compared to the win rates of the reds$''$ (obtained in Step 2). Figure 5.3 shows Step 3 graphically.

After Step 3, we have (a) the win rates of the reds$''$ in the two-versus-two scenarios, and (b) the win rates of the reds$_0$ in the same scenarios. By comparing the win rates, we should be able to determine the *success of the transfer*, i.e., the extent to which the behaviour of the reds$''$ has improved over the behaviour of the reds$_0$ because of the transferred knowledge. In the next section, we treat determining the success of the transfer as an experiment. There, we also elaborate on the specific comparison that we will perform on the win rates (see Subsection 5.3.4).

5.3 Experimental setup

To determine the success of the transfer in our use case as it is outlined in Section 5.2 we designed an experiment. The experiment consists of automated simulations in LWACS. The capabilities of LWACS are presented in Appendix A. Below, we present the setup of the experiment in detail. The setup is divided into four parts: the red teams (i.e., the reds$'$, the reds$''$, and the reds$_0$) (Subsection 5.3.1), the blue team (Subsection 5.3.2), the independent and dependent variables (Subsection 5.3.3), and a description of our method of analysis, by which we determine the success of the transfer (Subsection 5.3.4).

5.3.1 Red teams

In the use case, there are three red teams: the reds$'$, the reds$''$, and the reds$_0$. Apart from the scenarios in which they operate (and thus build up and/or use their knowledge), the red teams are equal. Each of the red teams consists of two fighter jet CGFs, a lead and a wingman. The capabilities of the CGFs are described in Appendix A.2. The goal of each red team is to learn how to defeat the blue team in a selection of different scenarios (see Section 5.2).

Each of the red teams uses the DECENT method for team coordination. This method has been explained in Subsection 3.2.3. The lead and the wingman both learn by means of dynamic scripting. Each uses their own rulebase. The reward function used during learning is AA-REWARD. This function has been described in Chapter 4.

5.3.2 Blue team

Depending on the task, the blue team consists of either one CGF (see Section 5.2, *Step 1*) or two CGFS, i.e., a lead and a wingman (see Section 5.2, *Step 2* and *Step 3*). The capabilities of the CGFS are described in Appendix A.2. The goal of the blue team is not to be defeated by red. The behaviour of blue is governed by scripts (see Appendix A.3).

5.3.3 Independent and dependent variables

Based on the use case, we define two independent variables in the experiment. The first independent variable is whether knowledge is transferred to the red teams that operate in the two-versus-two scenarios. This is the case for the reds$''$ (see Section 5.2, *Step 2*), but not for the reds$_0$ (see Section 5.2, *Step 3*). The second independent variable consists of the four two-versus-two scenarios (i.e., basic, close range, evasive, and lead-trail) for which we gather the win rates. Rather than averaging the win rates over the four scenarios, we are interested to see if any changes in performance caused by the transfer of knowledge to the reds$''$ vary between the four scenarios. The combination of these independent variables results in a 2×4 fully factorial design with eight conditions. The win rates are the dependent variable in the experiment.

5.3.4 Method of analysis

In our analysis of the results, we aim to measure the success of the transfer by comparing the win rates of the reds$''$ to the win rates of the reds$_0$. We apply three measures to the win rates in order to perform a meaningful comparison. The measures are (1) the initial performance measure, (2) the final performance measure, and (3) the turning point measure.

The initial performance measure calculates the mean win rate at the first encounter in the learning process. This measure captures how well the knowledge that was built up by the reds$_0$ can be directly applied by the reds$''$ in the two-versus-two scenarios before any further learning is allowed to take place.

In Chapters 3 and 4, we used the final performance measure and the turning point measure to analyse the performance of the learning CGFS. The two measures are explained in Subsection 3.3.6. Earlier, we have defined the final performance as the mean performance over the last 50 encounters. For the remainder of this chapter we redefine the final performance measure to be the mean performance of the last 30 encounters. By taking into account fewer encounters, we expect the final performance measure to more accurately reflect the stabilised performance after learning has taken place. The turning point in the learning process is the encounter at which point a moving window of 10 encounters contains more encounters that were won than encounters that were lost.

By use of the three measures, we are now able to compare the performance of the reds$''$ to the performance of the reds$_0$ in three areas. We perform the actual comparison by means of an ANOVA on the results of each of the three measures. The ANOVAS will show whether the transfer of knowledge leads to significantly better performance in the two-versus-two scenarios.

5.4 Experimental results

In this section, we present the results of the experiment. We begin by presenting the win rates of the reds$'$ in the two-versus-one scenario (Subsection 5.4.1). The win rates of the reds$'$ are an intermediary result, obtained by performing Step 1 in the use case (see Subsection 5.2.2). Next, we present the win rates of both the reds$''$ (Step 2) and the reds$_0$ (Step 3) in the two-versus-two scenarios (Subsection 5.4.2). Finally, we present the results of applying the three measures for the success of the transfer (Subsection 5.4.3).

5.4.1 Win rates of the reds$'$

In this subsection, we present and briefly discuss the win rates that are achieved by the reds$'$ in the two-versus-one scenario. Figure 5.4 shows the win rate of the reds$'$. The win rate starts at .407 at the first encounter, and then rises to an average win rate of .644 over the last 30 encounters. As can be seen in Figure 5.4, the win rate grows mildly but steadily over the course of the encounters. The win rate is the average of 150 runs of encounters, with each run consisting of 150 encounters in which the reds$'$ engaged the blue opponent. It is only after this number of encounters that the trend in the win rate became clearly visible, and that it became apparent that the win rate would not grow any further.

5.4.2 Win rates of the reds$''$ and the reds$_0$

In this subsection, we present the win rates achieved by both (a) the reds$''$ and (b) the reds$_0$ in the two-versus-two scenarios. Figure 5.5 shows the win rates. As in Subsection 5.4.1, the win rates are the average of 150 runs of encounters, where each run consists of 150 encounters (so, in total, 22,500 encounters). Here, we make two observations. The first observation is that the win rates of the reds$''$ and the reds$_0$ are clearly separated to some extent. The separation is prevalent in the basic and close range scenarios. Here, the win rate of the reds$''$ is clearly higher than that of the reds$_0$. The win rates converge after around 60 encounters. The second observation is that in the basic and close range scenarios, the win rate of the reds$''$ does not seem to improve over time. However, in the evasive and lead-trail scenarios, some improvement is visible in the win rates of both the reds$''$ and the reds$_0$.

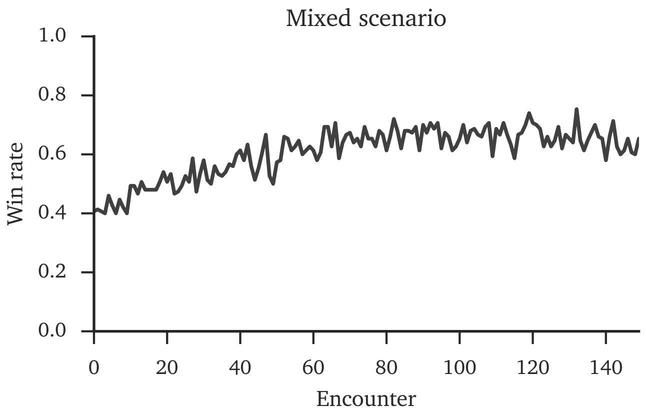


Figure 5.4 The win rates achieved by the reds′ in the two-versus-one mixed scenario.

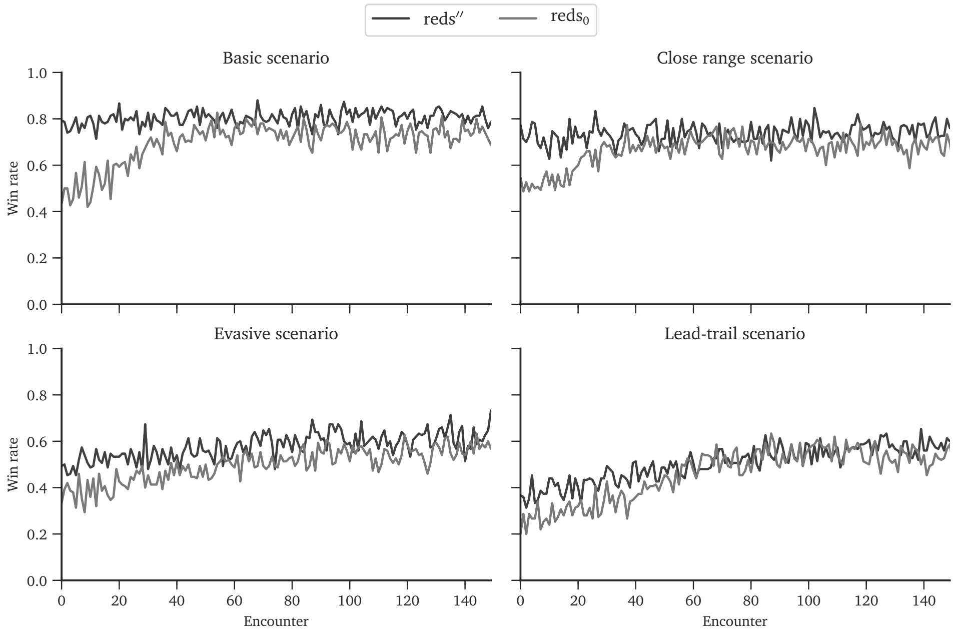


Figure 5.5 The win rates achieved by the reds″ and the reds$_0$ in the two-versus-two scenarios: (1) the basic scenario, (2) the close range scenario, (3) the evasive scenario, and (4) the lead-trail scenario.

5.4.3 Application of the three measures

In this subsection, we present the results of applying the three measures for the success of the transfer to the win rates of the reds$''$ and the reds$_0$: (A) the initial performance, (B) the final performance, and (C) the turning points. Furthermore, we apply an ANOVA to test for significant differences in the results of each measure.

A: Initial performance

First, we applied the initial performance measure to the win rates of the reds$''$ and the reds$_0$. Table 5.1 shows the results of applying the measure. In Table 5.1, we see that the initial performance of the reds$''$ is higher than that of the reds$_0$ in each of the four scenarios. The results indicate that the knowledge that is transferred from the reds$'$ to the reds$''$ provides an immediately observable benefit to the reds$''$. However, in the lead-trail scenario the difference in initial performance (0.003) appears to be somewhat negligible.

Table 5.1 The initial performance of the reds$''$ and the reds$_0$. A higher initial performance is better.

	Basic scenario		Close range scenario		Evasive scenario		Lead-trail scenario		Grand mean	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
reds$''$	.820	.385	.740	.440	.467	.501	.460	.500	.622	.457
reds$_0$	.556	.498	.616	.488	.252	.435	.457	.500	.470	.480

A two-way ANOVA was conducted on the influence of the two independent variables (transfer condition, scenario) on the initial performance of the reds$''$ and the reds$_0$. All effects were statistically significant at the $\alpha = .01$ level. The main effect of transfer condition yielded an F ratio of $F(1, 1192) = 29.857$, $p < 0.001$, indicating a significant difference between the initial performances of the reds$''$ and the reds$_0$. The main effect of scenario yielded an F ratio of $F(3, 1192) = 36.491$, $p < 0.001$, indicating a significant difference in the initial performance in each of the four scenarios. Furthermore, the interaction between the two independent variables was found to be statistically significant, $F(3, 1192) = 4.467$, $p < 0.01$. We performed a post hoc Tukey HSD test (see, e.g., Holmes et al., 2016) to determine the significant differences in initial performance between specific pairs of scenarios. The post hoc test revealed that the initial performance differed significantly between all scenarios, $p < 0.05$, except between the basic and close range scenarios.

B: Final performance

Second, we applied the final performance measure to the win rates of the reds$''$ and the reds$_0$. Table 5.2 shows the results of applying the measure. In each of the four scenarios, the reds$''$ reach a higher final performance than the reds_0 do. This means that the reds$''$ learn more effective behaviour in the scenarios. We observed a similar pattern for the initial performance (see A). However, in the case of the lead-trail scenario, the difference in initial performance was relatively small (0.003). Now, for the final performance, the difference has grown somewhat to 0.044.

Table 5.2 The final performance of the reds$''$ and the reds_0. A higher final performance is better.

	Basic scenario		Close range scenario		Evasive scenario		Lead-trail scenario		Grand mean	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
reds$''$	.819	.074	.734	.084	.553	.093	.498	.111	.651	.091
reds_0	.731	.116	.652	.093	.439	.116	.454	.141	.569	.117

A two-way ANOVA was conducted on the influence of the two independent variables (transfer condition, scenario) on the final performance of the reds$''$ and the reds_0. All effects were statistically significant at the $\alpha = .01$ level. The main effect of transfer condition yielded an F ratio of $F(1, 1192) = 182.092$, $p < 0.001$, indicating a significant difference between the final performances of the reds$''$ and the reds_0. The main effect of scenario yielded an F ratio of $F(3, 1192) = 582.882$, $p < 0.001$, indicating a significant difference in the final performance in each of the four scenarios. Furthermore, the interaction between the two independent variables was found to be statistically significant, $F(3, 1192) = 5.685$, $p <= 0.001$. A post hoc Tukey HSD test revealed that the final performance differed significantly between all scenarios, $p < 0.001$, except between the evasive and lead-trail scenarios.

C: Turning points

Third, we applied the turning point measure to the win rates of the reds$''$ and the reds_0. Table 5.3 shows the results of applying the measure. In each of the four scenarios, the reds$''$ reach lower turning points than the reds_0 do. This means that the reds$''$ learn more efficiently (viz. learn more effective behaviour in less encounters). The turning points in the evasive and the lead-trail scenarios show the largest differences (19.5 and 20.8, respectively). Thus, the turning points indicate that the knowledge that is transferred from the reds$'$ to the reds$''$ help the reds$''$ to efficiently learn to defeat the blue two-ships in the evasive and the lead-trail scenarios.

A two-way ANOVA was conducted on the influence of the two independent variables (transfer condition, scenario) on the turning points of the reds$''$ and the reds_0. All effects were statistically significant at the $\alpha = .01$ level. The main effect of transfer condition yielded an F ratio of $F(1, 1192) = 90.943$, $p < 0.001$, indicating a significant difference between the turning points of the reds$''$ and the reds_0. The main effect of scenario yielded an F ratio of $F(3, 1192) = 80.719$, $p < 0.001$, indicating a significant difference in the turning points in each of the four scenarios. Furthermore, the interaction between the two independent variables was found to be statistically

Table 5.3 The turning points of red. Lower turning points are better.

	Basic scenario		Close range scenario		Evasive scenario		Lead-trail scenario		Grand mean	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
reds$''$	10.4	1.6	10.8	2.4	16.4	9.8	21.6	19.6	14.8	8.4
reds$_0$	12.1	5.1	12.9	5.7	35.9	21.5	42.4	46.8	25.8	19.8

significant, $F(3, 1192) = 20.993$, $p <= 0.001$. A post hoc Tukey HSD test revealed that the turning points differed significantly between all scenarios, $p < 0.01$, except between the basic and close range scenarios.

5.5 Discussion

In this section, we discuss the results of the experiment. We cover three topics. First, we determine the success of the transfer (Subsection 5.5.1). Second, we discuss the performance of the reds$''$ in the new, unseen lead-trail scenario. Third, we briefly review the stationary win rates in the basic and close range scenarios (Subsection 5.5.3).

5.5.1 Success of the transfer

The results show clearly and consistently that the reds$''$ outperform the reds$_0$. As we defined in our use case (see Section 5.2), the common goal of the reds$''$ and the reds$_0$ was to defeat two blue opponents in two-versus-two scenarios. However, the reds$''$ received a *transfer of knowledge* of the knowledge built by the reds$'$ in a two-versus-one scenario.

In Subsection 5.3.4, we defined three measures for the success of the transfer: (a) the initial performance measure, (b) the final performance measure, and (c) the turning point measure. The application of these measures to the win rates of the reds$''$ and the reds$_0$ shows that:

(a) the transferred knowledge provides an immediate advantage to the reds$''$ in defeating the blue two-ship, before any learning by the reds$''$ takes place,

(b) the transferred knowledge enables the reds$''$ to learn more effective behaviour than the reds$_0$ throughout the encounters with the blue two-ship, and

(c) because of the transferred knowledge, the reds$''$ require less time than the reds$_0$ to start winning over 50% of the encounters with the blues.

Based on these results, we may conclude that the transfer of knowledge as outlined by our use case is to a large extent successful. Of course, for practical reasons our use case only includes a narrow selection of air combat scenarios. However, since our findings are consistent in each of the four two-versus-two scenarios, we may expect that the success of the transfer will generalise to some extent to other scenarios as well.

5.5.2 Improved performance in the lead-trail scenario

Out of the four two-versus-two scenarios in the use case, the lead-trail scenario is perhaps most interesting. In this scenario, the blues use a tactic that is not represented in the two-versus-one scenario. Therefore, this tactic is also not represented in the knowledge that is transferred from the reds$'$ to the reds$''$. Thus, the lead-trail scenario allows us to determine how well the transferred knowledge generalises to scenarios in which the opponents use new, unseen tactics.

In the results, we see that the initial and final performance of the reds$''$ are only slightly improved by the transferred knowledge (see Table 5.1 and Table 5.2, respectively). However, compared to the reds$_0$, the turning points of the reds$''$ are reduced by nearly 50% (from 42.4 to 21.6). So, in the case of the new, unseen tactic, the transferred knowledge does not appear to increase the effectiveness of the behaviour of the reds$''$, while it does increase the speed by which they find effective behaviour against the tactic.

In a survey of the rulebases of the reds$'$ (i.e., the knowledge that was transferred to the reds$''$), we observed that (a) the red lead had assigned a high weight to a particular evasive rule (i.e., evade incoming missiles by turning 180 degrees), but also that (b) the lead and the wingman had not converged to a role division for firing missiles. Therefore, we suspect that the combination of (a) the lead's preference for this particular evasive rule and (b) any firing rules in the rulebases of the two-ship with weights higher than the starting weights were sufficient to kick-start the learning process of the reds$''$.

Our results are in line with other works studying transfer learning in reinforcement learning applications. For instance, Spector and Belongie (2018) studied transfer learning in an application involving the automated playing of a simple Atari-like game. They report an increase in learning speed of 50 times in the case with transfer over the case without transfer.

5.5.3 Stationary win rates

In Figure 5.5, we see upward trends in most of the win rates. However, the win rates of the reds$''$ in (a) the basic scenario and (b) the close range scenario do not show an upward or downward trend. Instead, they appear to remain stationary around 0.8. This indicates that some form of optimum has been found in the weights in the rulebases of the reds$''$. Here, one of two situations is possible. On the one hand, the reds$''$ may be just successful enough to maintain the weights in the rulebase, without being forced to try new combinations of rules in the scripts. On the other hand, the stationary win rates might be the highest possible win rates that can be achieved in the scenarios using the rules that the reds$''$ (and thus also the reds$_0$) have available in their rulebases. An exhaustive search of the win rates that are achieved by all possible scripts in the basic and close range scenarios might indicate which of the two situations is currently at hand.

Still, if the reds$''$ no longer improve their behaviour, is there any benefit of the transfer of knowledge for the basic and close range scenarios? Despite the win rates of the reds$''$ remaining stationary, they also remain above the win rates of the reds$_0$. It takes nearly forty encounters for the win rates of the reds$_0$ to approach that of the reds$''$. This shows that overall, the transfer of knowledge leads to a better performance. However, the stationary win rates of the reds$''$ may also indicate that given blue's behaviour in these scenarios, the learning problem becomes easier when a second opponent is added. In essence, the reds may have been able to collect more reward (i.e., fire missiles with a higher P_k) *because* of the addition of a second, possibly easy-to-hit target. Further research should point out whether this causal relation actually exists.

5.6 Answering research question 3

In this chapter, we investigated the transfer of knowledge between CGFs. Specifically, we addressed research question 3.

Research question 3 reads: *To what extent can knowledge built with dynamic scripting be transferred successfully between CGFs in different scenarios?* To answer this question, we designed and implemented a use case for transfer learning in air combat simulations (Section 5.2). The use case consists of three steps. In Step 1 of the use case, a two-ship of red CGFs (which we call the reds$'$) engages a blue opponent in a two-versus-one scenario. In Step 2, the knowledge built up by this two-ship is transferred to a second two-ship (the reds$''$), who then use the knowledge to learn how to defeat two blue opponents in four different two-versus-two scenarios. In Step 3, a third two-ship (the reds$_0$) also learns how to defeat the blue two-ship in the four two-versus-two scenarios. However, they do so tabula rasa, viz. without a transfer of knowledge.

We used three measures to determine the success of the transfer: (a) the initial performance measure, (b) the final performance measure, and (c) the turning point measure (Section 5.3). The reds$''$, using the transferred knowledge, reach significantly higher performance than the reds$_0$ did, on each of the three measures. Even in the lead-trail two-versus-two scenario (which the reds$'$ had not seen, and thus could not transfer any knowledge of to the reds$''$), the transferred knowledge allowed the reds$''$ to learn more efficiently than the reds$_0$.

In conclusion, we answer research question 3 as follows. Based on the results of the simulations as outlined in the use case, we may conclude that we have to a large extent successfully transferred knowledge between air combat CGFs in different scenarios. Because air combat simulations often share common elements, we expect that the success of the transfer may extend beyond the scope of our use case as well.

6 A validation procedure for generated air combat behaviour models

In this chapter, we investigate research question 4. This research question reads: *How should we validate machine-generated air combat behaviour models for use in training simulations?*

Validation is an important step in the development of behaviour models, since it provides a structured way to determine whether the models are useful with regards to their intended purpose. However, there is no *one-size-fits-all* solution to the validation of behaviour models. Many different validation methods are available, each with their own strengths and weaknesses. It is up to the developer of the behaviour models to consider which validation methods are best applied.

We begin this chapter by briefly reviewing the available literature on (1) the validation of behaviour models and (2) the validation methods (Section 6.1). Next, we introduce new terminology (Section 6.2) tuned to the behaviour models designed for groups of four CGFS. These models and their validation are the subject of this chapter. Therefore, we design a validation process in a step-by-step manner (Section 6.3). Subsequently, we describe two specific elements of the validation process in detail. These elements are (1) the novel Assessment Tool for Air Combat CGFS (ATACC) which is presented in Section 6.4, and (2) the statistical analysis that is performed on the results of the ATACC, which is described in Section 6.5. Then, we present the steps for implementing the validation process (Section 6.6). Finally, we conclude the chapter by answering research question 4 (Section 6.7).

This chapter is based on the following publication.

- A. Toubman (2019). Validating Air Combat Behaviour Models for Adaptive Training of Teams. In: *Adaptive Instructional Systems*. Ed. by R. A. Sottilare and J. Schwarz. Springer International Publishing, pp. 557–571. DOI: `10.1007/978-3-030-22341-0_44`

6.1 Validating behaviour models

Since the advent of the use of simulation in military training (see, e.g., Sargent, 1939) there has been a rising interest in the *validation*[1] of simulation models (cf. Sargent, 2011; Kim, Jeong, Oh and Jang, 2015). Many definitions of validation have been stated throughout the literature (cf. Petty, 2010; Birta and Arbez, 2013; Bruzzone and Massei, 2017). When military simulations are discussed in particular, references are made to the definition of validation that is used by the US Department of Defense (2009). We use this definition from now onwards. For convenience, we restate the definition.

Definition 6.1 (Validation). Validation is ”[t]he process of determining the degree to which a model or simulation and its associated data are an accurate representation of the real world from the perspective of the intended uses of the model” (US Department of Defense, 2009).

The definition names four important concepts. The concepts are (1) a process, (2) a degree of accuracy, (3) a model (or simulation), and (4) the intended use of the model. We can readily fill in concepts (3) and (4). Regarding concept (3), the models that we wish to validate are newly generated behaviour models. Furthermore, regarding concept (4), the intended use of these models is to produce behaviour for opponent CGFs in air combat training simulations. However, this leaves open two questions for us to investigate: (1) what does the process precisely entail?; and (2) how should we determine the accuracy of the models? We discuss the two questions in Subsection 6.1.1 and Subsection 6.1.2, respectively. Subsection 6.1.3 concludes the section and provides an outlook on the remainder of the chapter.

6.1.1 What does the validation process precisely entail?

First, we investigate the question of *what the process precisely entails*. There is no *one-size-fits-all* solution for validation processes, since all different models have (1) different intended uses, and (2) different *associated works* available for use in the validation. Here, we use the notion of associated work to refer to a range of results of works performed, e.g., (1) baseline models, (2) expected output data, (3) conceptual diagrams of the modelled phenomenon, or (4) expert knowledge. This being so, we still observe that the various validation methods to be applied are well described in the literature. Petty (2010) names four types of validation methods for behaviour models: (1) informal methods, (2) static methods, (3) dynamic methods, and (4) formal methods. Below, we briefly describe these four validation methods, and provide examples of each. The descriptions and the examples are based on (Balci, 1994; Petty, 2010; Sargent, 2011).

[1]Validation is often paired with the related concept of *verification*. Whereas validation tries to answer the question *did we build the right model?*, the question that verification tries to answer is *did we build the model right?* We informally verified the generated models in Chapters 3, 4, and 5 by measuring their performance in automated simulations. The validation procedure that we design in this chapter is intended for determining whether the generated models are suitable for human-in-the-loop simulations.

Type 1: Informal methods. Informal methods are (mostly) qualitative methods that rely on subjective evaluations by subject matter experts of (1) the model or (2) associated works. Examples of informal methods are (a) inspection, (b) face validation, and (c) the Turing test.

Type 2: Static methods. Static methods evaluate (1) the structure of the model and (2) the flow of data within the model, both without executing the model. Examples of static methods are (a) data analysis, and (b) cause-effect graphing.

Type 3: Dynamic methods. Dynamic methods execute the model and evaluate the output that is produced by the model. Examples of dynamic methods are (a) sensitivity analysis, (b) predictive validation, (c) comparison testing, (d) regression analysis, and (e) hypothesis testing.

Type 4: Formal methods. Formal methods are methods that are based on mathematical proofs of correctness. According to both Balci (1994) and Petty (2010), formal methods provide (1) the most reliable conclusions of all validation methods, but at the same time are (2) the most difficult methods to apply to complex models. Examples of formal methods are (a) inductive assertions, and (b) predicate calculus.

An important factor in the choice of validation method(s) to use is the availability of associated works (Petty, 2010; Sargent, 2011). For example, dynamic methods can only be applied if (1) it is possible to execute the model with input that is relevant with regard to the intended use of the model, (2) data can be collected on the execution of the model, and (3) it is known how the collected data should be interpreted (e.g., compared to another available set of data). In other words, the choice of validation methods is always limited by practical considerations.

6.1.2 How should we determine the accuracy of the models?

The second question we would like to investigate reads: *how should we determine the accuracy of the models?* For instance, for a physics-based model, the accuracy of the model can be defined in terms of the number of faults that is allowed when the data that the model produces is compared to data that is measured in the real world. However, for behaviour models the question is particularly difficult to answer, since the notion of fault is difficult to grasp (see, e.g., Hahn, 2013; Hahn, 2017). Goerger, McGinnis and Darken (2005) identify five causes to the difficulty of validating behaviour models in general. Four[2] of these causes relate to the problem of defining the accuracy of a behaviour model. These four causes are: (1) the cognitive processes that are modelled may be nonlinear, which makes the processes as well as their models hard to reason about, (2) it is impossible to investigate all possible interactions that may arise in simulations

[2]The fifth cause is the lack of a standard validation process, which we discussed in Subsection 6.1.1.

because of the large number of interdependent variables in the models, (3) the metrics for measuring accuracy are inadequate, (4) there is no "robust"[3] set of input data for the models.

An important consequence of the difficulty of validating behaviour models is that the outcome of a validation should not be interpreted as either "the model is valid" or "the model is not valid", as it is practically impossible to "completely validate" a model (Birta and Arbez, 2013). Therefore, Birta and Arbez (2013) note that "degrees of success must be recognized and accepted." For them, it is important that the chosen validation methods are able to adequately reflect on the extent of the validity of the models.

6.1.3 Section conclusion and outlook

In summary, it is impossible to have a straightforward, general validation of behaviour models. Therefore, in the remainder of this chapter, we design a validation procedure that is tailored to (a) the generated behaviour models that we wish to validate (see Chapters 3 to 5), and also (b) the application (viz. training simulations) for which the behaviour models are intended (see Chapter 1). In the design, we consider (1) the associated works that are available, (2) the expert knowledge that may be applied, and (3) the measurement of degrees of accuracy of the models.

Looking forward, our validation procedure will consist of many interlocking parts (see Section 6.3). It is our opinion that the description of each part in the procedure must be accompanied by a comprehensive rationale behind each part. The reason should be *trust* in the validation process. Ultimately, validation is a matter of trust, i.e., establishing the trust that behaviour models are suitable for their intended application. Therefore, if the rationale behind one part of the validation process cannot be trusted, the wrong conclusions could be drawn from the results of the process. We acknowledge that the rationales provided in this chapter make the chapter quite lengthy and somewhat abstract. Still, we believe that these rationales are essential for appreciating the actual validation of newly generated behaviour models (i.e., the implementation of the validation process), which we perform in Chapter 7.

6.2 Terminology

In the previous chapters, we have mostly considered *two-ships* of CGFs. However, the human-in-the-loop simulations that we will discuss in this chapter (as well as in the next chapter) are designed to accommodate four human participants. In the simulations, the human participants are opposed by a team of four CGFs. Therefore, we now introduce the term *four-ship* to refer to such a team.

The larger team size requires us to rethink the manner by which we will discuss the behaviour models that produce the behaviour for the CGFs in a four-ship. So far, our experience has been that the behaviour models for the CGFs in a four-ship are treated as a single model. In particular,

[3]We interpret Goerger et al.'s (2005) use of "robust" here as "exhaustive".

when these behaviour models are designed by professionals, the behaviour models are carefully tuned to each other. So, they usually provide the illusion of a cohesive team at work. For this reason (being a cohesive team), we henceforth consider the four behaviour models that together control the behaviour of a four-ship to be an indivisible unit. For convenience, we introduce the term *4-model* to refer to the behaviour models of a four-ship. We define this term below.

Definition 6.2 (4-model). A 4-model is a combination of four behaviour models, which together are used to control the behaviour of a four-ship of air combat CGFs.

Using the term 4-model, we are now able to make a distinction between (1) 4-models that have been written by the professionals, and (2) 4-models that have been generated by means of machine learning. We introduce the terms *4P-model* (where the P stands for *professional*) and *4M-model* (where the M stands for *machine learning*) to refer to these two kinds of 4-model, respectively. We define these terms below.

Definition 6.3 (4P-model). A 4P-model is a 4-model that is written by professionals.

Definition 6.4 (4M-model). A 4M-model is a 4-model that is generated by means of machine learning.

6.3 Designing a validation process

In this section, we design a validation process for the validation of air combat CGF behaviour models. We do so along five design steps.[4] These design steps are: (1) outlining the process, (2) adding a baseline, (3) obtaining behaviour traces in human-in-the-loop simulations, (4) assessing the behaviour traces, and (5) equivalence testing. Below, we describe each of the five design steps and the rationale behind them.

Design step 1: Outlining the process. As the first design step, we draw the outline of the validation process. Figure 6.1 shows the outline. The validation process is placed in the middle of the figure. To the left of the process are the 4M-models that we wish to validate. Therefore, the 4M-models are the input to the validation process. The output of the validation process is the extent of the validity of the 4M-models (right).

Design step 2: Adding a baseline. The subjects of the validation (i.e., the 4M-models) are by themselves not sufficient input for the validation process. As Petty (2010) stated succinctly, validation "[is a] process[] that compare[s] things." Therefore, we require either (1) a

[4]The five design steps that we present in this section are an idealised abstraction of the design of our validation process. This abstraction is presented for the reader's convenience. In reality, the design was a demanding fuzzy optimisation task that required careful balancing of (1) the objective that we were trying to reach, and (2) the resources (both digital and human) that were available to us.

baseline model, (2) a set of expected output data, or (3) implicit expert knowledge as a reference to compare against the 4M-models.

For complex air combat behaviour models, it is almost infeasible to compile a set of expected output data, since the output depends on a wide range of possible interactions with other entities.[5] However, what we *do* have available are behaviour models that have been written previously by professionals (i.e., 4P-models). These 4P-models constitute a *sample* of all behaviour models that have been written by the professionals. The sample is in some sense comparable (see below) to the sample of 4M-models that have been generated by machine learning. Furthermore, we argue that since the 4P-models have been developed by means of the behaviour modelling process (see Section 2.1), the 4P-models have been validated to some extent. As a second design step, we therefore add 4P-models as the second input to the validation process (see Figure 6.2, highlighted).

Design step 3: Obtaining behaviour traces in human-in-the-loop simulations. Currently, a comparison of the 4M-models to the 4P-models in a meaningful way is a hard problem because of the aforementioned dependency on a wide range of input. We are unable to accurately predict if the models will produce comparable behaviour purely by inspecting the models. Therefore, as the third design step, we provide the models with the necessary input. We do so by submitting the models to human-in-the-loop simulations.

In the simulations, human pilots provide realistic input to the models, meaning that the behaviour of the pilots makes sense in the context of the training simulations for which the models are intended. Furthermore, by letting human pilots engage CGFs in simulations, we are able to obtain a sample of *behaviour traces*, i.e., recordings of the behaviour that the CGFs display. Figure 6.3 shows the composition of this design step. The human-in-the-loop simulations and the pilots are highlighted. The behaviour traces (not shown) serve as input to the remainder of the validation process.

Design step 4: Assessment of the behaviour traces. As the fourth design step, we aim to summarise the behaviour that is encoded in the behaviour traces into values that are (1) meaningful and (2) comparable between the 4M-models and the 4P-models. We do so by a structured form of face validation, which is one of the informal validation methods.

However, there is little to no information available on measures for CGF behaviour that are relevant to training simulations.[6] Therefore, in this design step, we make use of the

[5]The solution to this objection is using scenario models. However, well-balanced, adequate scenario models are beyond the scope of our research. Still, we use a similar idea by introducing the use of behaviour models which are written by professionals.

[6]An idea that was put forward at an early iteration of the design was to measure the improvement in skills of the *human* pilots after training in simulations with CGFs with 4M-models, in contrast to 4P-models. However, this idea brought along new problems, such as (1) selecting the right task to train in simulations, (2) choosing the right measures for the performance of the humans, and (3) using an appropriate training schedule.

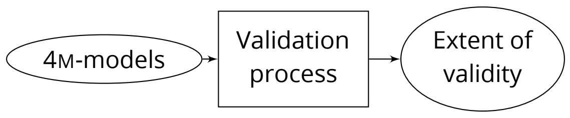


Figure 6.1 Design step 1. The outline of the validation process.

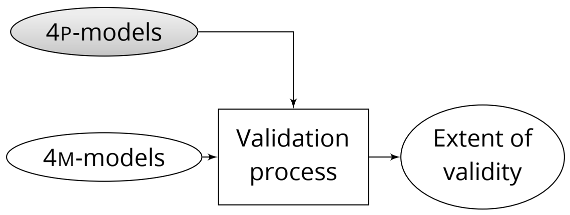


Figure 6.2 Design step 2. The 4P-models are added as a baseline.

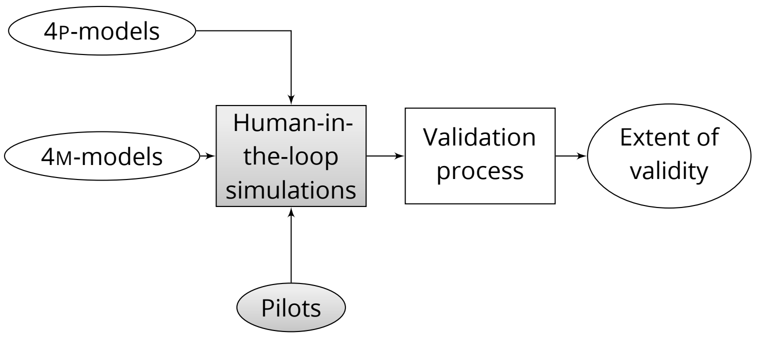


Figure 6.3 Design step 3. The 4M-models and the 4P-models are executed in human-in-the-loop simulations with the participation of human pilots.

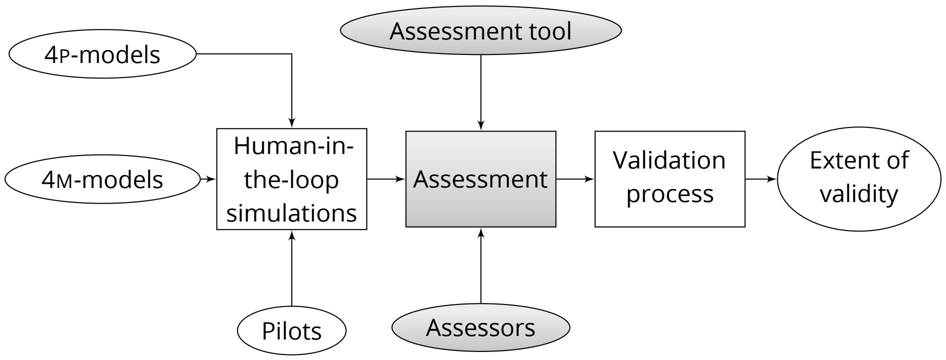


Figure 6.4 Design step 4. The results of the human-in-the-loop simulations are subjected to assessments by assessors that make use of an assessment tool.

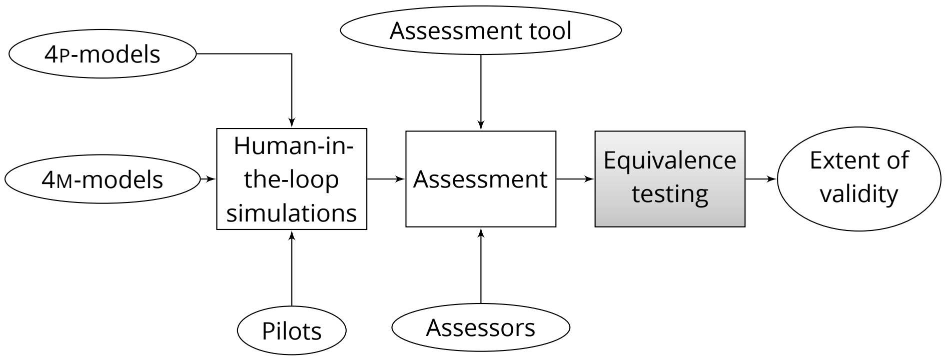


Figure 6.5 Design step 5. The results of the assessments are analysed by means of equivalence testing.

implicit knowledge of expert evaluators. We leverage this knowledge in two manners. First, we elicit knowledge on measures for behaviour of air combat CGFs, and then structure this knowledge into an assessment tool (see Section 6.4). This tool enables a structured assessment of CGF behaviour. Second, expert evaluators review the behaviour traces that we have collected, and then assess the behaviour that the CGFs display. The assessments are performed by means of the newly developed assessment tool.

The use of expert evaluators as assessors relates back to the question of *how should we determine the accuracy of the models*. Since we are unable (as of yet) to codify the measures for the accuracy of CGFs behaviour in a manner that is (1) complete and (2) objective, the main source of these measures is the implicit knowledge of expert evaluators. Sadagic (2010) used a similar validation method in a similar context (i.e., the behaviour of urban warfare CGFs).

Figure 6.4 shows the addition of (1) the assessment (centre, highlighted), including (2) the assessors (bottom, highlighted) and (3) the assessment tool (top, highlighted) to the validation procedure.

Design step 5: Equivalence testing. At this point in the validation process, we have two sets of data: (1) the assessments of the 4P-models, and (2) the assessments of the 4M-models. We wish to compare these two sets of data in a meaningful way. Since we used the 4P-models as the baseline, we assume that the assessments of the 4P-models contain information about the desirable properties of air combat CGF behaviour. Based on this assumption, we define the following measure of validity of the 4M-models.

Definition 6.5 (Measure of validity of the 4M-models)**.** The 4M-models are valid to the extent that (1) the assessments of the 4M-models and (2) the assessments of the 4P-models can be measured to be equivalent.

Obviously, a simple comparison (viz. determining if the difference between the assessments equals zero) of the assessments is too strict. The results of our assessments include noise from multiple sources (e.g., the pilots in the human-in-the-loop simulations, and bias of the assessors). Furthermore, standard statistical significance tests do not suffice, since these tests check for differences rather than for equivalence. We found a solution in a form of comparison testing that is called *equivalence testing*. We further describe the equivalence testing in Section 6.5.

Figure 6.5 shows the result of this design step. We replace the remainder of the validation process by equivalence testing (highlighted). The output of the equivalence testing is the extent of the validity of the 4M-models.

In summary, we have designed a validation procedure by means of which behaviour models for CGFs may be validated. In the procedure, we use two validation methods: (1) face validity in a structured form by means of an assessment tool, and (2) comparison testing between the assessment results of the 4P-models and the 4M-models. However, two gaps remain in the procedure. The first gap is the assessment tool by which the assessors can assess the behaviour that the models produce in a structured manner. We develop this tool in Section 6.4. The second gap is the comparison testing that is performed on the results of the assessments. We describe the comparison testing in Section 6.5. Afterwards, in Section 6.6, we provide a step-by-step procedure for implementing the validation procedure.

6.4 The Assessment Tool for Air Combat CGFs

In this section, we present the Assessment Tool for Air Combat CGFs (ATACC). Below, we first describe the development of the ATACC. Next, we look at its implementation.

We consulted four former instructor pilots for the development of a novel assessment tool. During multiple brainstorming sessions, we identified (1) an appropriate format for the tool, and (2) the specific behaviour that we wished to measure with the tool.

The assessment of behaviour is a major topic of research in the fields of (1) behavioural sciences and (2) human resource management (cf. DeNisi and Murphy, 2017). For this reason, we performed a literature review in order to find formats which could be used as a basis for our assessment tool. The review guided us towards the tool known as the behaviourally anchored rating scale (BARS) (Debnath, Lee and Tandon, 2015).

A BARS (plural: BARSS) is a scale that is intended to measure specific *performance dimensions* (Snell, Morris and Bohlander, 2015, p. 321). In order to aid the assessors who use the BARS in identifying the behaviours, the levels of the scale are marked with anchors. These anchors consist of *critical incidents*, e.g., objectively observable behaviours that are (un)desirable in the performance dimensions. We refer to the work by Phillips, Shafer, Ross, Cox and Shadrick (2006) for an example of BARSS for tactical behaviour in the military domain.

Together with the instructor fighter pilots, we identified three performance dimensions that should be taken into consideration in the assessment of the behaviour of air combat CGFS. These performance dimensions are (1) the *challenge* provided by the CGFS, (2) the *situational awareness* that the CGFS display, and (3) the *realism* of the behaviour of the CGFS. Below, we briefly describe the three performance dimensions.

Performance dimension 1: Challenge. The tool should measure whether (1) the CGFS behave in such a way that the human participants in the simulations need to think about and adjust their actions, and (2) whether the CGFS provide some form of *training value* to the simulations.

Performance dimension 2: Situational awareness. The tool should measure whether (1) the CGFS appear to sense and react to changes in their environment, and (2) whether multiple CGFS belonging to the same team appear to acknowledge each other's presence.

Performance dimension 3: Realism. The tool should measure (1) whether the CGFS behave as can be expected from their real-world counterparts, and (2) whether the CGFS use the capabilities of their platform (including, e.g., sensors and weapons) in a realistic manner.

After the identification, we attempted to formulate examples of behaviour that relate to each of the performance dimensions. This was done in an iterative manner, so that examples that were proposed could be critically analysed by each of the instructor fighter pilots. We formulated eight examples of behaviour in total. Below, we list these eight examples of behaviour. In each of the examples, *red air* refers to the CGFS, whereas *blue air* refers to the human participants in the human-in-the-loop simulations. Four of the examples relate to performance dimension 1, *Challenge*.

Example of behaviour 1. Red air forced blue air to change their tactical plan.

Example of behaviour 2. Red air forced blue air to change their shot doctrine[7].

Example of behaviour 3. Red air was within factor range[8].

Example of behaviour 4. Blue air was able to fire without threat from red air.[9]

Subsequently, two examples relate to performance dimension 2, *Situational awareness*.

Example of behaviour 5. Red air acted on blue air's geometry.

Example of behaviour 6. Red air acted on blue air's weapon engagement zone[10].

The remaining two examples relate to performance dimension 3, *Realism*.

Example of behaviour 7. Red air flew with kinematic realism.

Example of behaviour 8. Red air's behaviour was intelligent.

Next, we attempted to define *critical incidents* based on the eight examples of behaviour. In other words, we tried to formulate desirable and undesirable instances of the examples of behaviour, that could be observed in an objective manner. The critical instances could then be placed as anchors on their respective performance dimensions in order to form the BARSs. However, despite our best efforts, we were unable to define satisfactory critical incidents that (1) objectively described situations that could be observed, and (2) once observed in a simulation, would indicate the performance of the CGFs in a performance dimension for the *entire* simulation. Consequently, we were unable to use the BARS format for our assessment tool.

Rather than abandoning the examples of behaviour that were formulated, we decided to substitute the BARS format by a related format. This format is the behaviour observation scale (BOS). In contrast to a BARS, a BOS defines examples of behaviour and attempt to measure the frequency of the occurrence of the examples (Snell et al., 2015, p. 321). Following the new way, rather than requiring predefined anchors to guide the assessors, it is the assessor who determines if a given behaviour is displayed, and if so, how often. Here, an appeal is made to the implicit expert knowledge that the assessor possesses on critical incidents that we are as of yet unable to explicitly define.

We created a new BOS for the assessment of air combat CGFs. In this BOS, we used the eight examples of behaviour that were defined earlier in this section. We attached a five-point Likert scale to each example of behaviour, to indicate that example's occurrence in a simulation: (1) never, (2) rarely, (3) sometimes, (4) often, or (5) always.

[7]Jargon: pre-briefed instructions for the use of air-to-air weapons.

[8]Jargon: the range within which opponents have to be taken into account in the selection of tactical actions.

[9]We formulated this behaviour from the viewpoint of blue air, since we were unable to satisfactorily state the behaviour from the viewpoint of red air.

[10]Jargon: the airspace in front of a fighter jet in which a fired missile can be effective.

In addition to the eight examples of behaviour, we added a ninth example. This example states on a high level the behaviour that we desire from the CGFs that are being assessed. The purpose of the ninth example is to capture the general opinion on the suitability of the behaviour of CGFs . Therefore, this example functions as a sort of *control item* on the BOS. Below, we state the ninth example of behaviour.

Example of behaviour 9. Red air's behaviour tested blue air's tactical air combat skills.

The ninth example of behaviour is also rated using a five-point Likert scale, but with different options than the first eight examples: (1) strongly disagree, (2) disagree, (3) undecided, (4) agree, or (5) strongly agree.

6.5 Equivalence testing

We incorporate Schuirmann's (1987) two one-sided t-tests (TOST) method in the validation process to determine the equivalence of (1) the responses given on the ATACC for 4P-models, and (2) the responses given on the ATACC for 4M-models. The TOST method involves the application of two one-sided t-tests. They should calculate to what extent two measured means do not differ from each other, given a margin of error that is called the *indifference zone*. We briefly introduce the TOST method below (Subsection 6.5.1). Next, we explain how we use the TOST to measure the *extent* of the validity of the models that are the subject of the validation (Subsection 6.5.2).

6.5.1 Equivalence testing with TOST

The TOST method tests for equivalence of the means of two populations (cf. Meyners, 2012; Anderson-Cook and Borror, 2016; Lakens, 2017). This means that the method (1) starts with the assumption that two populations are different, and then (2) collects evidence to show that the populations are the same. Note that this is the opposite of traditional tests that compare two populations (e.g., Student's t-test), which (1) start with the assumption that two populations are similar or even the same, and then (2) collect evidence to show that the populations are different.

In TOST, the assumption that two populations are different (viz. the *null hypothesis* or H_0) is stated as follows.

$$H_0: \quad \mu_A - \mu_B \leq \delta_L \quad \text{or} \quad \mu_A - \mu_B \geq \delta_U \tag{6.1}$$

Here, the difference of the means of two populations A and B are compared. Two populations are considered different if the difference of their means lies outside of the indifference zone $[\delta_L, \delta_U]$. For the remainder of this chapter, we assume that the indifference zone is symmetrical, i.e., $\delta = \delta_U = -\delta_L$. However, we are interested in examining the alternative hypothesis (or

H_1) that the means are *not* different, i.e., the difference between the means lies inside of the indifference zone. Following from H_0, we formulate H_1 as follows.

$$H_1: \quad \delta_L < \mu_A - \mu_B < \delta_U \tag{6.2}$$

If the TOST finds evidence that the difference of the means lies within the indifference zone under the assumption that it does not, we reject H_0 and do not reject H_1, meaning that we conclude that the populations are the same (up to a very small difference). Finding this evidence is done by splitting H_0 into two hypotheses which can be tested using standard one-sided t-tests. The p-value of the TOST then becomes the maximum of the two p-values that are obtained from the two one-sided t-tests.

The outcome of the TOST greatly depends on the value chosen for δ. Until recently, δ could not be calculated directly. It was either (1) prescribed by regulatory agencies (e.g., in the field of pharmacology) or (2) determined by subject matter experts based on reference studies or expectations about the data (e.g., in psychology) (cf. Meyners, 2012; Anderson-Cook and Borror, 2016; Lakens, 2017). For our validation, it is difficult to determine a suitable δ, since we have neither a regulatory agency, nor a reference study available. However, in 2016, an objective calculation of δ was introduced by Juzek (2016). The calculation of this delta δ (henceforth: Juzek's δ) is as follows.

$$\delta = 4.58 \frac{s_p}{N_p} \tag{6.3}$$

Here, s_p is the pooled standard deviation in the two samples under comparison, and N_p is the pooled number of data points in the samples. Juzek found the coefficient (4.58) by simulating a large number of TOST applications. The coefficient was approximated in such a way that Juzek's δ gives the TOST the appropriate statistical power ($1-\alpha = 95\%$, $1-\beta = 80\%$).

6.5.2 Measuring an extent of validity

As mentioned in Section 6.1, the validation process should not produce an absolute outcome. Rather, the process should reflect *degrees of success*, i.e., the extent to which models can be said to be valid. Although we have selected the TOST method for our equivalency tests, we have not yet defined how the results of the TOST should be interpreted to arrive at a judgement on the validation of the 4M-models.

The TOST provides us with a test of equivalence of the assessments for each example of behaviour. In other words, nine tests of equivalence are performed in total to compare the ATACC assessments of the 4M-models to the ATACC assessments of the 4P-models. Therefore, we propose that we measure the extent of the validity of the 4M-models along the number of equivalences that are found by the TOST.

6.6 Implementing the validation process

In this section, we briefly state a step-by-step procedure that can be followed to implement the validation process that was described in this chapter. The procedure consists of five steps. We describe these steps below.

Step 1. Defining the baseline. We collect a set of 4P-models to serve as the baseline. The size of this set is a trade-off between (1) the 4P-models that are available to use, and (2) the number of 4P-models that can be practically used in human-in-the-loop simulations, so that after the generation of 4M-models (see Step 2) sufficient behaviour traces per 4P-model can be (a) collected (see Step 3) and (b) assessed (see Step 4).

Step 2. Generating models by means of machine learning. We generate a set of 4M-models by means of dynamic scripting. These 4M-models are the subject of the validation. Here, the same trade-off on the size of the set of 4P-models (see Step 1) holds for the size of the set of 4M-models.

Step 3. Human-in-the-loop simulations. The 4P-models and the 4M-models are used to control the behaviour of a four-ship of CGFS in human-in-the-loop simulations. In the simulations, the CGFS are opposed by a four-ship that is controlled by human participants. The behaviour that both the CGFS and the human participants show is recorded as behaviour traces that can be reviewed at a later time.

Step 4. Assessments. Subject matter experts assess the behaviour traces that were obtained from the human-in-the-loop simulations. The assessments are performed by means of the ATACC.

Step 5. Equivalence testing. We perform equivalence tests to compare (1) the behaviour produced by the 4P-models to (2) the behaviour produced by the 4M-models.[11] The results of the equivalence tests indicate to what extent the 4M-models are valid for use in training simulations.

6.7 Answering research question 4

In this chapter, we addressed research question 4. This research question reads: *How should we validate machine-generated air combat behaviour models for use in training simulations?* To answer this question, we investigated the validation methods that are available in the literature (Section 6.1). Next, we defined new terminology (Section 6.2) that allows us to refer concisely

[11]In the future, the validation procedure presented in this chapter may be adapted to compare the behaviour of 4M-models that have been generated using different machine learning techniques, such as deep learning as it has been applied in the ALPHAGO program (see, e.g., Silver et al., 2016, 2017b).

to the combined behaviour models of a four-ship of CGFS. With the use of the new terminology, we designed a validation process for the validation of behaviour models for air combat CGFS (Section 6.3).

The validation process has two important features. The first feature is the use of a novel assessment tool for the assessment of the behaviour that CGFS display in human-in-the-loop simulations (Section 6.4). The second feature is the use of equivalence testing, a form of hypothesis testing that determines whether two sets of data may be considered equivalent (Section 6.5). In the validation process, equivalence testing is used to determine whether the behaviour that is produced by generated models is assessed as equivalent to the behaviour that is produced by models that are written by professionals. Finally, we summarised the implementation of the validation process, including (1) the use of the ATACC and (2) the equivalence testing by means of TOST, into a step-by-step procedure (Section 6.6). This procedure forms the answer to research question 4.

7 Validation of generated behaviour models in training simulations

In this chapter, we investigate research question 5: *To what extent are air combat behaviour models generated by means of dynamic scripting valid for use in training simulations?*

To answer this question, we implement the steps of the validation procedure that we presented in Chapter 6. Since we designed the validation procedure with five steps, we cover all steps in five sections (from Section 7.1 to Section 7.5) as follows. First, we describe the four 4P-models that together act as the baseline of the validation (Section 7.1). Second, we describe the 4M-models that together act as the subject of the validation (Section 7.2). Next, we discuss the human-in-the-loop simulations that are performed with the help of human F-16 pilots (Section 7.3). These pilots engage four-ships of CGFs that use the 4-models, in a manner that resembles the operation of training simulations. The behaviour of the CGFs in these simulations are assessed by a group of expert assessors (Section 7.4). We present the results of the assessments, including the equivalence tests that are performed (Section 7.5). Additionally, we discuss the results and our interpretation of the validity of the generated 4M-models (Section 7.6). Finally, we conclude this chapter by answering research question 5 (Section 7.7).

This chapter is based on the following publications.

- A. Toubman, J. J. Roessingh, P. Spronck, A. Plaat and H. J. Van den Herik (2016b). Rapid Adaptation of Air Combat Behaviour. In: *ECAI 2016 - 22nd European Conference on Artificial Intelligence*. Ed. by G. A. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum and F. Van Harmelen. Vol. 285. Frontiers in Artificial Intelligence and Applications. The Hague, The Netherlands: IOS Press, pp. 1791–1796. DOI: 10.3233/978-1-61499-672-9-1791

- A. Toubman (2019). Validating Air Combat Behaviour Models for Adaptive Training of Teams. In: *Adaptive Instructional Systems*. Ed. by R. A. Sottilare and J. Schwarz. Springer International Publishing, pp. 557–571. DOI: 10.1007/978-3-030-22341-0_44

7.1 Defining the baseline: The 4P-models

We had to obtain four 4P-models that were written by professionals for use in training simulations. For this task we could rely on the work performed previously by a group of professionals (see Netherlands Aerospace Centre, 2017a) who had designed and worked out four 4P-models. These four 4P-models were inspected by us and considered to be fit for the task to be a sample that forms the baseline in the validation process (see Step 1, *Defining the baseline*, Section 6.6).

The differentiating factor between the four 4P-models was the starting formation of the involved CGFs. Each starting formation defines (1) the spatial configuration of the CGFs, and (2) their initial speeds. Therefore, the starting formation is an important factor in the interactions between the CGFs and the human participants in human-in-the-loop simulations. We refer to the four starting formations as F_1, F_2, F_3, and F_4. We return to the four starting formations in Section 7.2, where they are used in the 4M-models that are generated.

The 4P-models were modelled by the professionals in the SMART BANDITS[1] behaviour modelling program (Netherlands Aerospace Centre, 2017a). As such, the 4P-models were in the form of finite-state machines (FSMS). As a modelling technique, FSMS allow behaviour modellers to organise behaviour into different states, only one of which can be active at the same time (cf. Adam, Taillandier and Dugdale, 2017; Yildiz, Akcal, Hostas, Ure and Inalhan, 2018). Based on observations by the CGF that uses the behaviour model, the CGF enters a certain state in the model, and then only executes the behaviour belonging to that state. Models in the form of FSMS are easily displayed in a graphical manner, in contrast to, e.g., scripts. Especially as the number of rules in a script grows, it becomes difficult for the behaviour modeller to keep track of the possible interactions between the conditions and consequences of all rules in the script. Instead, FSMS clearly indicate which transitions between states are possible, and when these transitions are made.

7.2 Generating behaviour models: The 4M-models

Next, we had to generate four new 4M-models. We did so by means of machine learning, in the form of the dynamic scripting technique. This sample of 4M-models is the subject of the validation (see Step 2, *Generating models by means of machine learning*, Section 6.6).

Our goal was to generate a "counterpart" 4M-model to each of the four 4P-models. To generate these counterparts, we formulated two requirements for the 4M-models. First, we required that each 4M-model should use the same starting formation (either F_1, F_2, F_3, or F_4) as its counterpart 4P-model. The reasoning for using the same starting formations is that we viewed these starting formations as an essential part of the training simulations. Furthermore, reusing the same starting formations was a manner of forcing dynamic scripting to work within the same constraints as the

[1]The SMART BANDITS program is introduced in Appendix D as part of the Fighter 4-Ship simulator.

professionals do, when modelling the behaviour of the CGFs. Therefore, pairing each 4P-model with a counterpart 4M-model allows for a fair comparison between the modelling capabilities of the professionals and dynamic scripting.

As the second requirement for the counterparts, we required the 4M-model counterparts to use the same modelling technique as the 4P-models. In pretests when we trained ourselves with the problem at hand, we detected an important difference between (a) the behaviour that is produced by scripts such as generated by dynamic scripting, and (b) the behaviour that is produced by FSMs such as created in SMART BANDITS. In a direct comparison, the behaviour produced by the scripts appeared to be erratic and indecisive at certain moments during the simulations. We attribute this indecisiveness to unforeseen interactions between some rules and specific observations made by the CGFs. As an example, consider the case where a red CGF is simultaneously (a) attacking a blue CGF, as well as (b) being attacked by another blue CGF. Due to small changes in the movements of the two blue CGFs affecting the firing of red's offensive and defensive rules, the red CGF would appear to oscillate between (a) continuing to attack the first blue CGF, and (b) defending against the attack of the second blue CGF. This behaviour, while possibly (and unexpectedly) quite effective in automated simulations, is very *unhumanlike* and therefore unacceptable in a real-world human-in-the-loop simulation.

Rather than attempting to augment the rules to prevent this behaviour, we decided to modify the dynamic scripting algorithm, enabling it to generate FSMs. We will elaborate on the reason for generating FSMs by means of dynamic scripting in Appendix E. In this appendix, we also describe the modifications that we made to the dynamic scripting algorithm. In brief, we divided FSMs into their constituent states and transitions, and then treated these states and transitions as rules for use in dynamic scripting's rulebase. For simplicity, we continue to use the term *rules* and *rulebase* in the remainder of this chapter.

Below, we further discuss how the 4M-models were generated. First, we briefly discuss the origin of the rules that were used in the rulebases of the CGFs (Subsection 7.2.1). Thereafter, we describe the automated simulations by means of which we generated the 4M-models that later acted as the subjects of the validation (Subsection 7.2.2).

7.2.1 The rules in the rulebases

Because dynamic scripting requires a rulebase with rules in order to generate a behaviour model, we had to consider an appropriate source for the rules. We chose to derive the rules from the four 4P-models that we had available (see Section 7.1). To do so, we divided the 4P-models into their constituent states and transitions. From these states and transitions, we extracted any states and transitions related to the starting formations of the CGFs. Then, we removed any duplicates. The remaining rules formed the rulebase to be used by dynamic scripting.

Because dynamic scripting only recombines rules, and does not synthesise any new rules, the algorithm could only generate FSMs that closely resembled the 4P-model. Therefore, we augmented the rulebase with rules that we call variant rules. Each variant rule was based on one of the rules that already existed in the rulebase. In each variant rule, we made one or more small changes compared to the rule on which the variant was based, in terms of altered values such as (but not limited to) headings, time-outs, and sensor readings. For example, if a state directed the CGF to turn 90 degrees, we added also a variant of that state which directed the CGF to turn -90 degrees. The rationale behind the altered values was to choose values that were (1) sensible (e.g., not firing all missiles at once) and (2) meaningful (viz. rather adding a few variants with large changes in values, than adding many variants with small changes in values). The variant rules were added to the rulebase alongside the existing rules.

Additionally, during the translation of the states and transitions to the rules, we discovered that transitions leading from one state to another were tightly coupled to the states from which the transitions originated. Therefore, rather than implementing each transition as a separate rule, we embedded each transition into the rule that defined the state from which the transition originated.

So, we created sixteen copies of the rulebase. These copies formed four groups of four rulebases. Each group of four rulebases served as a starting point for one of the four 4M-models that were generated (see Subsection 7.2.2). Finally, we assigned one of the four starting formations to each of the four groups, and then added this starting formation as a rule to each rulebase in that group.

7.2.2 Automated simulations

In the end, we generated four 4M-models by means of automated simulations. In this subsection, we describe the strategy by which we did so. The strategy, which we refer to as the generation strategy, consists of three steps. Figure 7.1 shows the three steps graphically. Below, we discuss the three steps of our generation strategy.

Step 1. Four red CGFs engaged four blue CGFs in simulated air-to-air combat encounters. The reds learned by means of dynamic scripting, making use of a group of four rulebases (see Subsection 7.2.1). The reds approached the blues in the starting formation as was programmed in their rulebases. The blues were scripted to approach the reds as described in their own starting formation, which we call starting formation A. Once the blues detected the reds, the blues were scripted to attack the reds, only interrupting their attack to perform defensive manoeuvres if the blues were under attack themselves. The reds were allowed to learn over the course of 40 encounters. They coordinated their actions by means of the DECENT coordination method. Each encounter ended when either (a) each CGF in a team had been hit by a missile from the opposing team, or (b) a time limit of ten minutes was reached. We applied BIN-REWARD[2] as the reward function for the red team: the red team

[2]We preferred the use of AA-REWARD here. However, at the time, we were unable to correctly implement it in the STAGE simulation environment.

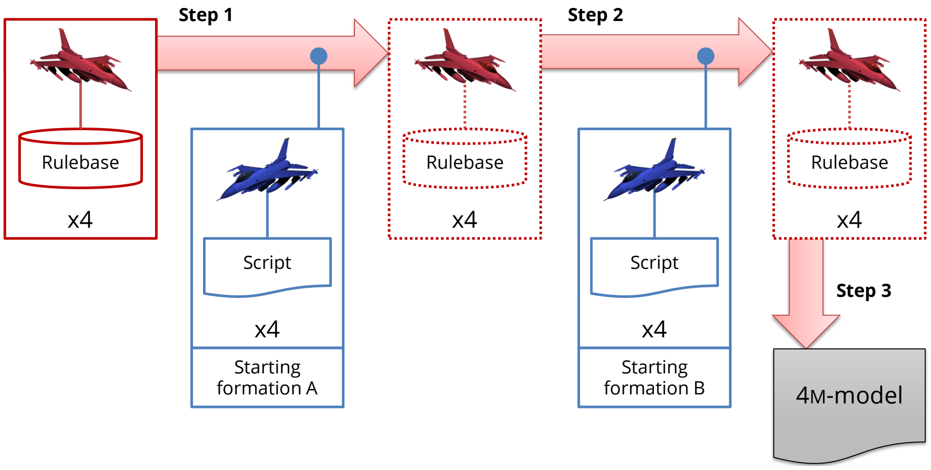


Figure 7.1 The three steps of the generation strategy. Step 1: the reds learn to defeat a four-ship of blues (which use starting formation A) over the course of 40 encounters. After these encounters, the rulebases (shown with dotted lines) are optimised towards defeating blues that use starting formation A. Step 2: the same reds (viz. using the same rulebase and the weights therein) learn to defeat a different four-ship of blues (which use starting formation B) over the course of 40 encounters. After these encounters, the rulebases (again shown with dotted lines) are optimised towards defeating blues that use either starting formation A or B. Step 3: rules are extracted from the rulebases to create a 4M-model.

was awarded a reward value of 1 if the blue team was defeated, and a reward value of 0 otherwise.

Step 2. We took the simulation from Step 1, and replaced the starting formation of the blues by a new starting formation, called starting formation B. The remainder of the simulation was left unchanged, including the (weights in) the rulebases of the reds. Thereby, we essentially transferred the reds and their knowledge (see Chapter 5) from the simulation in Step 1, to the simulation in Step 2. Next, the reds were allowed to learn to defeat the blues with starting formation B over the course of 40 encounters.

Step 3. We formed a script out of the rules of each rulebase. We did so in the following manner. First, we divided the rulebase into groups. Each group contained one of the original rules, plus its variants (see Subsection 7.2.1). Next, out of each of these groups of rules, we selected the rule with the highest weight for inclusion in the script. In case of a tie between the weights of two rules, a rule was selected at random. This way, we ensured that the rules in the resulting script together formed a complete and functional FSM.

By applying the generation strategy, we obtained four scripts. These scripts together form a single 4M-model. We repeated the generation strategy for each of the four 4M-models that we wished to generate. This resulted in the four 4M-models that were the counterparts to the four 4P-models.

The simulations described in this section were performed in the STAGE simulation environment, which is part of the Fighter 4-Ship simulator (see Appendix D). To allow CGFS in STAGE to learn by means of dynamic scripting, we implemented the dynamic scripting algorithm in the form of a new program. We call this program STAGEDS. STAGEDS used the application programming interface (API) of STAGE to control (a) the CGFS in the simulations, as well as (b) the simulations themselves (viz. starting, stopping, and restarting the simulations) in order to automate the simulations as required for the learning process of the red CGFS.

7.3 Human-in-the-loop simulations

We use human-in-the-loop simulations to determine how a four-ship of red CGFS behaves when the CGFS interact with human participants (see Step 3, *Human-in-the-loop simulations*, Section 6.6). The simulations were performed in the Fighter 4-Ship simulator.

The behaviour of the reds was controlled by means of eight 4-models: the four 4P-models (see Section 7.1) plus the four 4M-models (see Section 7.2). Using these eight 4-models, we defined eight *scenarios*. Each scenario was a simulation configuration in which a four-ship of red CGFS approached the human participants from the simulated north. In each scenario, the red four-ship used either (a) one of the four 4P-models or (b) one of the four 4M-models, so that each of the 4-models was used in one of the scenarios.

The human participants in the simulations were active-duty Royal Netherlands Air Force (RNLAF) F-16 pilots from Volkel Airbase (all male, n = 16, age $\mu = 32.0$, $\sigma = 5.35$), and one former RNLAF F-16 pilot (age = 60).[3] No selection criteria were applied. The active-duty pilots were assigned to the human-in-the-loop simulations based on availability. Experience levels ranged from *wingman* to *weapons instructor pilot*.

Over the course of three days, five teams of four participants controlled the blue CGFS in the Fighter 4-Ship. Before the simulations took place, the participants received a “mission briefing” document that described (1) the capabilities of the blue CGFS that they would control, and (2) the capabilities of the red CGFS that the participants were to expect in the simulator. The eight scenarios were presented sequentially in a random order. The participants were unaware of the origin of the 4-models controlling the red CGFS (i.e., the simulations were performed in a

[3]One of the active-duty participants had to leave after four scenarios. This situation presented us with three options: (1) continue without this participant (viz. with a three-ship), (2) cancel the remaining simulations, or (3) substitute the participant with a former F-16 pilot who was available. Since the participant had a non-commanding role in the four-ship, we deemed his influence in the decision-making of the human participants to be minimal. Still, by controlling the fourth blue CGF, he provided valuable input that allowed the red CGFS to function. Furthermore, participants were scarce. We decided that the collection of data was paramount, and let the former F-16 pilot (mentioned above) substitute the participant in the remaining simulations.

single-blinded fashion). Each scenario ended when either all four red CGFs, or all four human participants were defeated.

The human-in-the-loop simulations were recorded using Personal Computer Debriefing System (PCDS). These recordings included (1) the voice communication that took place among the human participants, and (2) video recordings of the multi-functional displays (MFDS) of the ships occupied by the human participants. In total, 33 recordings[4] were stored.

7.4 Behaviour assessments

The behaviour that the reds displayed in the human-in-the-loop simulations were assessed by human experts (see Step 4, *Assessments*, Section 6.6). Active-duty RNLAF F-16 pilots from Leeuwarden Airbase acted as assessors (all male, $n = 5$, age $\mu = 35.2$, $\sigma = 5.17$). Assessors were selected on having *tactical instructor pilot* or *weapons instructor pilot* qualification. We considered either of these qualifications to be sufficient in order to function as the *training specialist* that the validation criterion calls for. All five assessors had the weapons instructor pilot qualification.

The assessments were performed by means of the ATACC. We implemented the rating items of the ATACC as a single-page paper form. In our implementation, we made three additions to the rating items: (1) we added a field for the *tactical* (i.e., the code name) of the assessors for later reference, (2) we added a field for the *operational status* of the assessors to gain insight into their experience level, and (3) we added two fields for indicating the specific recorded encounter that was viewed by the assessor. The form is presented as it was used in the behaviour assessments in Appendix F.

Originally, we had planned to let each assessor assess all of the 33 recordings within a three hour time span. However, a pilot study with two weapons instructor pilots (not counted above) revealed that this was infeasible. We subsequently reduced the pool of recordings available for rating to 16 recordings. These 16 recordings came from two teams that completed all eight scenarios (i.e., simulations with the four 4P-models and the four 4M-models) in human-in-the-loop simulations. From this reduced pool of recordings, we assigned ten recordings to each rater, consisting of (1) eight recordings from one of the two teams in random order, and (2) two recordings from the other team. Furthermore, the weapons instructor pilots in the pilot study expressed that they were unable to adequately assess the intelligence of the red CGFs (rating item 8) and the extent to which the red CGFs tested the skills of the pilots in the simulator (rating item 9) without knowing the experience levels of these pilots. Based on this feedback, we made the decision to disclose the experience levels to the assessors during the assessments.

For the assessments, the assessors were provided with (1) a laptop computer with mouse and headphones, (2) a stack of ten ATACCs, and (3) an instruction sheet. The PCDS recordings were

[4]Two teams were not available to complete all eight scenarios. Together, these two teams completed nine scenarios: the eight scenarios, plus one duplicate.

opened on the computer. Each ATACC was marked with a unique code that referred to a specific recording in PCDS. The assessors were instructed to view the recordings in the order as indicated by their ATACCS.

We planned two analyses on the responses to the ATACC: (1) *equivalence testing* on the responses to the ATACC, and (2) calculating of the *inter-rater reliability*. We briefly describe them below.

Equivalence testing. We apply a method known as TOST (cf. Meyners, 2012; Anderson-Cook and Borror, 2016; Lakens, 2017) on the responses to the ATACC to determine the extent of the validity of the 4M-models. Equivalence testing is part of the validation procedure (see Step 5, *Equivalence testing*, Section 6.6).

Inter-rater reliability. We calculate the intraclass correlation (ICC) as a measure of inter-rater reliability, viz. how consistently recordings are rated between assessors. We did not include the calculation of the ICC in the validation procedure. However, since the number of assessors in our validation is limited, the ICC serves as an indication of the trustworthiness of the assessments.

7.5 Results of the behaviour assessments

A summary of the responses to the ATACC is given in Table 7.1. The responses to the Likert scale rating items were coded as integer values ranging from 1 (Never/Strongly disagree) to 5 (Always/Strongly agree). The coding for rating item four (*Blue air was able to fire without threat from red air*) was inverted so that the values reflected the occurrence of red behaviour (i.e., red influencing blue's ability to fire). Below, we present the results of the equivalence tests (Subsection 7.5.1) and the inter-rater reliability analysis (Subsection 7.5.2). Furthermore, we include a brief review of feedback that was received from the assessors during the assessments (Subsection 7.5.3).

7.5.1 Equivalence testing

We applied Schuirmann's (1987) TOST method to determine the equivalence of (1) the responses given on the ATACC for 4P-models, and (2) the responses given on the ATACC for 4M-models. We calculated δ (as Juzek's δ) for the responses to each rating item of the ATACC, and then performed the TOST on the responses to each rating item. The TOST was performed using the `TOSTtwo.raw` function from R's `TOSTER` package, with Welch's t-test as the underlying one-sided test. We chose to use Welch's t-test here because of the unequal sample sizes.[5] The δ and the results of the TOST

[5]There is an ongoing discussion on the topic of whether parametric tests such as the t-test are suitable for use on ordinal Likert-scale data. Parametric tests have on multiple occasions been shown to be robust against violated assumptions (such as non-normal, ordinal data) (cf. Norman, 2010; De Winter, 2013; Derrick and White, 2017). Using parametric tests in our TOST allows us to use well-tested, publicly available tools such as the mentioned R package.

(t-value, degrees of freedom [*df*], p-value, and the 90% confidence interval (CI) of the difference of the means) are shown in Table 7.2. In Table 7.2, the bold p-values indicate a significant result of the TOST. Based on the results of the TOST, we may conclude that the responses to rating items 1, 2, 5, 7, 8, and 9 are equivalent between the 4P-models and the 4M-models.

The TOST did not find equivalence for rating items 3, 4, and 6. For these rating items, we conducted a follow-up test to determine if the responses to these rating items significantly differed between the 4P-models and the 4M-models. This follow-up test was a standard two-sided Welch's t-test. A significant difference was found for rating items 3 and 6. These two rating items read as *Red air was within factor range* (rating item 3), and *Red air acted on blue air's weapons engagement zone* (rating item 6). For both rating items, the responses indicated a higher frequency of the behaviour that was rated for the 4M-models (see Table 7.1). The remaining rating item read as *Blue air was able to fire without threat from red air* (rating item 4). The responses to rating item 4 were neither significantly equivalent, nor significantly different. Therefore, we may conclude that their relationship is undecided.

7.5.2 Inter-rater reliability

An inter-rater reliability analysis was carried out on the nine rating items of the ATACC. The ICC estimate and its 95% CI were calculated using the `icc` function from R's `irr` package, based on a two-way random effects model (consistency, multiple raters/measurements) (cf. Koo and Li, 2016). The ICC estimate and its 95% CI are shown in Table 7.3. The reported values indicate moderate agreement between the assessors (Koo and Li, 2016).

7.5.3 Feedback on the assessments

The assessors that took part in the validation provided direct verbal feedback during and after the assessments. The feedback concerned both (a) the ATACC questionnaire, and (b) the simulations that were shown. Below, we briefly review the feedback that we received.

A general topic of feedback on the ATACC questionnaire was its reliance on the insight (or ”gut feeling”) of the assessors over quantifiable measures. Assessors noted that they were trained to deal with quantifiable measures, and as such on occasion they found it difficult to assess the behaviour of the CGFs along the rating items of the ATACC. However, the assessors also understood that if the behaviour could be defined completely in quantifiable terms, their insight would not have been required.

Rating item 7 (*Red air flew with kinematic realism*) was a frequent subject of comments by the assessors. This rating item was either called unclear, or the assessor stated that he did not have the means to assess the flying performance of the red CGFs. Table 7.1 shows that out of the nine rating items, the fewest responses were collected for this rating item.

One assessor commented that several manoeuvres that the CGFs (using 4M-models, unknown to the assessor) displayed were interesting from a training perspective, but also unrealistic

Table 7.1 Summary of the ATACC responses: the number of responses (n), mean response (μ), and standard deviation (σ) of the responses to the ATACC rating items for the 4P-models and the 4M-models. The highest means (viz. behaviours that were observed the most) and the lowest standard deviations (viz. the most agreement between the raters) are highlighted.

	4P-models			4M-models		
Rating item	n	μ	σ	n	μ	σ
1	28	3.04	0.79	24	3.25	0.99
2	28	2.07	0.98	24	2.33	1.13
3	28	3.18	1.19	24	**3.92**	1.02
4	27	2.26	0.86	24	2.71	0.91
5	28	3.29	0.71	24	3.42	**0.58**
6	28	2.75	0.89	24	3.33	0.70
7	22	**3.82**	**0.66**	20	3.70	0.73
8	28	2.86	0.80	24	2.96	0.69
9	27	3.81	0.68	24	3.63	0.65

Table 7.2 Results of the TOST method per rating item (r.i.). The TOST was based on Welch's t-test. For rating items where the TOST method did not find equivalence, an additional standard (Welch's) t-test was performed. Significant p-values at the $\alpha = 0.05$ level are indicated in bold. The relevance (rel.) of the outcome of the tests is indicated in the rightmost column.

	TOST					Standard t-test				
R.i.	δ	t	df	p	90% CI	t	df	p	95% CI	Rel.
1	0.798	2.322	43.9	**.012**	[−0.637, 0.208]					eq.
2	0.944	2.307	45.9	**.013**	[−0.758, 0.234]					eq.
3	1.000	0.855	50.0	.198	[−1.251, −0.225]	-2.41	50.0	**.020**	[−1.353, −0.124]	diff.
4	0.800	1.414	47.5	.082	[−0.866, −0.032]	-1.81	47.5	.077	[−0.949, 0.050]	und.
5	0.590	2.551	49.9	**.007**	[−0.432, 0.170]					eq.
6	0.725	0.643	49.7	.262	[−0.953, −0.214]	-2.64	49.7	**.011**	[−1.018, −0.149]	diff.
7	0.697	-2.674	38.5	**.005**	[−0.247, 0.483]					eq.
8	0.677	2.779	50.0	**.004**	[−0.448, 0.246]					eq.
9	0.604	-2.223	48.8	**.015**	[−0.122, 0.502]					eq.

eq. = equivalent, diff. = different, und. = undecided

Table 7.3 Results of the intraclass correlation analysis.

		F-test with true value 0			
ICC	95% CI	value	$df1$	$df2$	p
0.651	[0.494, 0.770]	2.86	63	252	.000

as the manoeuvre did not seem to provide a direct tactical advantage. This is an example of the creativity offered by machine learning. However, in particular this case, the creativity was detrimental to the realism of the behaviour. In the future, it may be possible to detect and filter out such manoeuvres from generated behaviour models.

7.6 Discussion

In this section, we discuss the results from the behaviour assessments. These results are the foundation on which we base our perception of the validity of the 4M-models. In our validation procedure, we defined the extent of the validity of our 4M-models as the extent to which these models were assessed as equivalent to the 4P-models that were obtained. Below, we cover the following five topics: our key finding (Subsection 7.6.1) and the context in which it should be interpreted (Subsection 7.6.2), the implications of the finding (Subsection 7.6.3), and the limitations of the study (Subsection 7.6.4).

7.6.1 Key finding

Our key finding is that out of the nine rating items of the ATACC, six items are assessed as equivalent between the 4M-models and the 4P-models by expert human assessors. Of the remaining three rating items, the responses to two rating items (i.e., rating items 3 and 6) were found to be statistically different between the 4P-models and the 4M-models, whereas the responses to one rating item (i.e., rating item 4) were found to be inconclusive. Although the responses to these three rating items do not directly support the validity of the 4M-models, the responses to rating items 3 and 6 indicate that the behaviour produced by the 4M-models was perceived as *more* challenging (rather than *less* challenging) than the behaviour produced by the 4P-models. Therefore, we nonetheless consider these responses to be a positive signal (and to some extent in support of our key finding) for the use of machine-generated behaviour models in training simulations.

As mentioned in Chapter 6, degrees of success in a validation study must be “recognized and accepted” (Birta and Arbez, 2013), since it is practically impossible to “completely validate” behaviour models. Although the ATACC can certainly be improved, we have successfully used it to demonstrate that machine learning is capable of generating behaviour models that are perceived as equivalent, at least on six out of the nine rating items, to behaviour models that have been manually written by professionals. We interpret these results as a moderately strong indicator for validity of the generated models regarding the application of training simulations.

7.6.2 Placing our key finding in context

In contrast to Chapters 3 to 5, which together demonstrated the problem-solving power of machine learning in automated simulations, this chapter addressed the application of a machine learning technique in a setting dominated by humans. We applied machine-generated behaviour models in human-in-the-loop simulations, and then worked with human assessors to assess the behaviour produced by the models. Furthermore, the simulations and the assessments that followed were only possible after consulting human subject matter experts on the best ways to assess the behaviour of opponent CGFs in training simulations, and then capturing this knowledge in the validation procedure (see Chapter 6). Clearly, applying machine learning in training simulations is as much a social challenge as it is a technical one.

In Chapter 6, we put a large amount of effort in critically considering each step of our validation procedure. We consider this to be a major strength of our validation study, aiding in both social and technical acceptance of our proposed use of machine learning. The inclusion of subject matter experts in the formulation of this procedure enables the procedure to focus on the envisioned usage of the behaviour models, viz. opponent behaviour in training simulations. For contrast, we briefly consider one of the few related studies that made use of human-in-the-loop simulations. Teng et al. (2013) applied both (a) adaptive and (b) non-adaptive machine-generated behaviour models in WVR air combat simulations involving human fighter pilots, with the goal of improving training simulations. The fighter pilots were presented with questionnaires on which they could assess six properties of the behaviour produced by the two kinds of models, i.e., to what extent the behaviour was perceived to be (1) predictable, (2) intelligent, (3) skillful, (4) challenging, (5) adaptive, and (6) aggressive. It is not mentioned why these specific properties were used, or what the desirable assessment scores would be for the intended application of the generated behaviour models. In our view, studies regarding the use of machine learning in training simulations would greatly benefit from involving training experts and other subject matter experts at an early stage, so that the research can be more focused on the potential added value to the training simulations, and therefore to the humans that depend on the simulations.

7.6.3 Implications

To the best of our knowledge, the validation study that we present in this chapter is the first of its kind in the context of BVR air combat training simulations, using behaviour models generated by means of machine learning. As such, an important step has been made in bringing a machine learning application to the area of military training simulations. Furthermore, the equivalence of the responses to six of the rating items show, if not complete validity, at least a large potential for the use of machine learning in this area. As a next step, the behaviour models that are currently available for use in training simulations could for instance be supplemented by machine-generated models, in order to simultaneously (1) provide more variation in the training, and (2) gather further experience with applying machine-generated models in a real-world training setting.

7.6.4 Limitations

While we have critically reviewed our validation procedure (see Chapter 6) and implemented it to the best of our abilities with the available resources (this chapter), two limitations affect our study. First, the ATACC questionnaire was only informally validated in preliminary simulations. Second, the number of assessors in the validation was limited. Both can be attributed to our attempt to optimise the use of limited resources. Disregarding these limitations somewhat, we hope that the results of our study may serve as an incentive for further research in this area, including, e.g., a refinement of the ATACC as a research instrument. Such a refinement should increase (a) the inter-rater reliability, and therefore also (b) the value of the assessments. One approach might be the inclusion of mission essential competencies (MECs) for human F-16 pilots (cf. Alliger, Beard, Bennett Jr, Symons and Colegrove, 2013; Tsifetakis and Kontogiannis, 2017) in the ATACC.

7.7 Answering research question 5

In this chapter, we investigated the validation of machine-generated behaviour models. Specifically, we addressed research question 5: *To what extent are air combat behaviour models generated by means of dynamic scripting valid for use in training simulations?*

To answer research question 5, we apply the validation procedure that we presented in Chapter 6. We generated behaviour models by means of dynamic scripting, and then used these behaviour models to control CGFs in human-in-the-loop simulations. Equivalence testing shows that on six of the nine rating items of the ATACC, the CGFs that are controlled by machine-generated behaviour models are rated equivalently to CGFs that are controlled by behaviour models written by professionals.

Answering the research question very precisely proves to be quite difficult. While we could, for instance, translate six out of nine rating items to 66.667 %, we consider such a percentage to be meaningless regarding validity. In our view, the results appear to *moderately* indicate validity, but the responses to the remaining three rating items do not support the notion of validity as we have defined it for ourselves. Therefore, we answer research question 5 as follows: the air combat behaviour models generated by means of dynamic scripting are *to a moderate extent* already valid for use in training simulations. In the future, this will certainly improve.

8 Conclusions

In this chapter we summarise our answers to the five research questions (Section 8.1) and formulate our answer to the problem statement posed in Chapter 1 (Section 8.2). Finally, we provide two recommendations for future research (Section 8.3).

8.1 Answers to the research questions

In Section 1.3, we posed five research questions. Below, we provide a summary of our answers to the five research questions based on the research performed in the previous chapters.

Research question 1: *To what extent can we generate air combat behaviour models that produce team coordination?*

The answer to the first research question is derived from Chapter 3. We were able to implement three methods within the rule-based framework of dynamic scripting (TACIT, CENT, DECENT) that each produced a form of team coordination. We demonstrated how the three methods lead to a flexible division of roles within a two-ship of air combat CGFS. Out of the three methods, the team coordination produced by the CENT method (viz. centralised coordination by means of communication) resulted in the most effective behaviour that reached the highest win rates. Our answer to the research question is that by means of dynamic scripting, we are able to (a) generate multiple forms of team behaviour, (b) compare the effectiveness of the produced behaviour, and (c) easily inspect the roles assumed by the team members.

Research question 2: *To what extent can we improve the reward function for air combat CGFS?*

The answer to the second research question is derived from Chapter 4. The common but simple binary reward function such as BIN-REWARD (i.e., 0 for losing an encounter, 1 for winning an encounter) offers rewards that are (a) sparse and (b) unstable. We developed two new reward functions: (1) DOMAIN-REWARD, which offers less sparse but still somewhat unstable rewards, and (2) AA-REWARD, which offers somewhat sparse rewards that are entirely stable. We found that DOMAIN-REWARD did not improve the performance of air combat CGFS over the use of BIN-REWARD, but the use of AA-REWARD lead to a performance increase of 12.6% while maintaining the same learning speed as BIN-REWARD.

Our answer to the research question is that we are able to improve the reward function by making the rewards offered by the reward function (a) less sparse and (b) stable. The improvements result in a 12.6% increase in final performance.

Research question 3: *To what extent can knowledge built with dynamic scripting be transferred successfully between CGFs in different scenarios?*

The answer to the third research question is derived from Chapter 5. To answer this research question, we designed a use case for transfer learning in air combat simulations. In the use case, two distinct two-ships learn to defeat two blue opponents in a set of two-versus-two scenarios. One of the two-ships uses transferred knowledge that has been previously built up in a two-versus-one scenario. In practical terms, dynamic scripting allows for very straightforward transfers of knowledge. Knowledge is stored in the form of the weights associated with each rule in a rulebase. Therefore, a transfer simply entails copying the rules and their weights to a new rulebase. We determined the success of the transfer by comparing the performance of the two two-ships (one with, and one without transferred knowledge) by three measures. Each of the three measures indicated that the use of the transferred knowledge resulted in a significant increase in performance. Thus, our answer to the research question is that knowledge built with dynamic scripting can be successfully transferred to a large extent.

Research question 4: *How should we validate machine-generated air combat behaviour models for use in training simulations?*

The answer to the fourth research question is derived from Chapter 6. There is no one-size-fits-all solution to the validation of machine-generated behaviour models. Therefore, our answer to the research question is a newly developed validation procedure consisting of five steps.

Step 1. Selecting a sample of professionally written behaviour models (the 4P-models) which exemplify desirable behaviour.

Step 2. Generating a sample of behaviour models by means of machine learning (the 4M-models).

Step 3. Applying the 4P-models and the 4M-models in human-in-the-loop simulations.

Step 4. Assessment of the behaviour produced by the behaviour models by means of the ATACC questionnaire.

Step 5. Equivalence testing to determine whether the assessments of the behaviour produced by the 4M-models are statistically equivalent to the assessments of the behaviour produced by the 4P-models.

In Step 5, assuming the assessments are statistically equivalent, we consider the 4M-models to be valid for use in human-in-the-loop simulations. To the best of our knowledge, this is the first time a validation procedure for machine-generated air combat behaviour models has been formulated and documented.

Research question 5: *To what extent are air combat behaviour models generated by means of dynamic scripting valid for use in training simulations?*

The answer to the fifth research question is derived from Chapter 7. We applied the validation procedure from Chapter 6. As the baseline, we selected 4P-models that were designed by subject matter experts. We generated new 4M-models, and then applied both the 4P-models and 4M-models in realistic human-in-the-loop F-16 fighter jet simulations. The assessment of the behaviour of the CGFs was carried out by active duty F-16 instructor pilots. On six out of the nine rating items on the ATACC questionnaire, the assessments were statistically equivalent between the 4P-models and the 4M-models. On two of the remaining rating items, the behaviour produced by the 4M-models was perceived as more challenging, which we consider to be a positive indicator for the capabilities of dynamic scripting in the air combat domain. Although we have clearly not completely validated the 4M-models in the context of the validation procedure, we must consider that (a) the majority of the assessments of the behaviour were statistically equivalent, and that (b) "degrees of success must be recognized and accepted" (Birta and Arbez, 2013). Therefore, our answer to the research question is that the 4M-models are valid to a moderate extent.

8.2 Answer to the problem statement

In this section we answer the problem statement that was posed in Section 1.3. Our answer is based on the answers to the five research questions discussed in the previous section.

Problem statement: *To what extent can we use dynamic scripting to generate air combat behaviour models for use in training simulations, in such a way that the five challenges of generating air combat behaviour models are met?*

In Chapter 1, we stated five challenges (A-E) that must be met by machine learning techniques in order for the techniques to be considered suitable for use in the air combat domain. Below, we briefly restate the five challenges and declare how dynamic scripting, in combination with the research presented in this thesis, meets each challenge.

Challenge A: Producing team coordination. The use of the CENT coordination method enables CGFs that learn by means of dynamic scripting to coordinate their behaviour and specialise into roles.

Challenge B: Computationally evaluating CGF behaviour. The AA-REWARD reward function evaluates the behaviour of air combat CGFs and provides stable rewards. These rewards lead to a performance increase over the conventional binary reward function.

Challenge c: Efficient reuse of acquired knowledge. We have shown that cgfs that learn by means of dynamic scripting are able to improve their performance in complex scenarios by reusing the knowledge that was built in simpler scenarios.

Challenge d: Validating generated behaviour models. We have developed a validation procedure, and applied this procedure to behaviour models generated by means of dynamic scripting. We have concluded that the models are valid for use in human-in-the-loop simulations to some extent, although there is room for improvement.

Challenge e: Generating accessible behaviour models. This challenge is met by the use of dynamic scripting, as the behaviour models produced by dynamic scripting are in the form of human-readable rules.

Based on our research, our answer to the problem statement is that dynamic scripting greatly facilitates the automatic generation of air combat behaviour models, while being flexible enough to be moulded into answers to the challenges. Challenges (a-c) are met by adapting (a) the rules, (b) the reward function, and (c) the manner in which we apply dynamic scripting algorithm to the air combat domain. Meanwhile, challenge e is met by the design of the dynamic scripting algorithm itself. Still, challenge d remains somewhat open. This challenge is perhaps the most critical one, as the validation of the generated models would mean that we, as machine learning researchers, are confident that the use of our models in training simulations will teach important skills and abilities to the air force pilots of the future. We look forward to the day that challenge d is completely met, but until then this challenge is a good reminder that machine learning can make a difference not only in simulations, but also in the real world.

8.3 Recommendations for future research

Based on the research performed in the thesis, we recommend two areas for future research. They are (1) refinement of the validation procedure, and (2) quantification of the training value of cgf behaviour.

Refinement of the validation procedure: In the thesis, we have developed and applied a validation procedure for air combat behaviour models (see Chapters 6 and 7). However, as we have mentioned, there is no one-size-fits-all solution to the validation of the models, and a large part of the design of the procedure consists of the experiences and opinions of subject matter experts. Therefore, collecting more of these experiences and opinions, and putting them to the test in an empirical manner, may lead to a procedure that has more power to establish the validity of behaviour models. In the future, the validation procedure may also investigate the accessibility of the behaviour models as perceived by the subject matter experts (see, e.g., Fürnkranz, Kliegr and Paulheim, 2018). Such an investigation will allow the comparison of the accessibility of the models that are produced by different machine learning techniques.

Quantification of the training value of CGFS behaviour: The ultimate goal of generating behaviour models for CGFS by means of machine learning is to increase the training value of human-in-the-loop simulations. To this end, it is important to identify how the behaviour of the CGFS influences the learning of air combat concepts by the trainees in the simulations. Once this knowledge can be captured in a reward function (see Chapter 4), it may be possible for the machine learning techniques to optimise the generated behaviour models for the training value for the trainees, rather than towards a concept (e.g., the P_k value of missiles) that act as a proxy for the training value.

References

Abbass, H., A. Bender, S. Gaidow and P. Whitbread (2011). Computational red teaming: Past, present and future. In: *IEEE Computational Intelligence Magazine* 6.1, pp. 30–42. DOI: 10.1109/mci.2010.939578.

Abbeel, P., A. Coates, M. Quigley and A. Y. Ng (2007). An application of reinforcement learning to aerobatic helicopter flight. In: *Advances in neural information processing systems*, pp. 1–8.

Abdellaoui, N., A. Taylor and G. Parkinson (2009). 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: RTO/NATO.

Adam, C., P. Taillandier and J. Dugdale (2017). Comparing Agent Architectures in Social Simulation: BDI Agents versus Finite-state Machines. In: *Proceedings of the 50th Hawaii International Conference on System Sciences (2017)*. Hawaii International Conference on System Sciences. DOI: 10.24251/hicss.2017.032.

Aleshire, P. (2005). *Eye of the Viper: The Making of an F-16 Pilot*. Lyons Press. ISBN: 9781599217222.

Alford, R., H. Borck, J. Karneeb and D. W. Aha (2015). Active Behavior Recognition in Beyond Visual Range Air Combat. In: *Proceedings of the Third Annual Conference on Advances in Cognitive Systems (ACS)*. Cognitive Systems Foundation. Atlanta, Georgia.

Alliger, G. M., R. Beard, W. Bennett Jr, S. Symons and C. Colegrove (2013). A psychometric examination of mission essential competency (MEC) measures used in air force distributed mission operations training needs analysis. In: *Military Psychology* 25.3, pp. 218–233.

Alpaydin, E. (2010). *Introduction to Machine Learning*. 2nd. The MIT Press. ISBN: 9780262012430.

Anderson-Cook, C. M. and C. M. Borror (2016). The difference between "equivalent" and "not different". In: *Quality Engineering* 28.3, pp. 249–262. DOI: 10.1080/08982112.2015.1079918.

Andrychowicz, M., F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel and W. Zaremba (2017). Hindsight experience replay. In: *Advances in Neural Information Processing Systems*, pp. 5048–5058.

Arulkumaran, K., M. P. Deisenroth, M. Brundage and A. A. Bharath (2017). A Brief Survey of Deep Reinforcement Learning. In: *arXiv preprint arXiv:1708.05866*.

Ausink, J. A., W. W. Taylor, J. H. Bigelow and K. Brancato (2011). *Investment Strategies for Improving Fifth-Generation Fighter Training*. Tech. rep. TR-871-AF. RAND Corporation.

Balci, O. (1994). Validation, verification, and testing techniques throughout the life cycle of a simulation study. In: *Annals of Operations Research* 53.1, pp. 121–173. DOI: 10.1109/wsc.1994.717129.

Banks, S. B. and M. R. Stytz (2003). Progress and Prospects for the Development of Computer-Generated Actors for Military Simulation: Part 2-Reasoning System Architectures and Human Behavior Modeling. In: *Presence* 12.4, pp. 422–436. ISSN: 1054-7460. DOI: 10.1162/105474603322391640.

Bellman, R. (1957). *Dynamic Programming*. Rand Corporation research study. Princeton University Press. ISBN: 9780691079516.

Besbes, O., Y. Gur and A. Zeevi (2014). Stochastic multi-armed-bandit problem with non-stationary rewards. In: *Advances in neural information processing systems*, pp. 199–207.

Bianchi, R. A., L. A. Celiberto Jr, P. E. Santos, J. P. Matsuura and R. L. de Mantaras (2015). Transferring knowledge as heuristics in reinforcement learning: A case-based approach. In: *Artificial Intelligence* 226, pp. 102–121. DOI: 10.1016/j.artint.2015.05.008.

Bigelow, J. H., W. W. Taylor, S. C. Moore and B. Thomas (2003). *Models of operational training in fighter squadrons*. Tech. rep. MR-1701-AF. RAND Corporation.

Bijlsma, F. (2014). Evolving dynamic AI opponents for OpenTTD using Dynamic Scripting and Grammatical Evolution. MA thesis. Utrecht University.

Birta, L. G. and G. Arbez (2013). *Modelling and simulation: exploring dynamic system behaviour*. Springer Science & Business Media. ISBN: 978-1-4471-2783-3.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc. ISBN: 0387310738.

Blizzard Entertainment (2010). *StarCraft II*. URL: http://eu.blizzard.com/en-gb/games/sc2/ (visited on 17/03/2019).

Bolton, A., K. P. Tucker, H. Priest, A. McLean, J. Beaubien, W. Stacy, S. Wiggins, R. Wray and J. Mooney (2016). Live, Virtual and Constructive Training Fidelity (LVC TF) Special Session. In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*. Vol. 60. 1. SAGE Publications Sage CA: Los Angeles, CA, pp. 2001–2004. DOI: 10.1177/1541931213601455.

Bongers, A. and J. L. Torres (2014). Technological change in U.S. jet fighter aircraft. In: *Research Policy* 43.9, pp. 1570–1581. ISSN: 0048-7333. DOI: 10.1016/j.respol.2014.03.009.

Borck, H., J. Karneeb, R. Alford and D. W. Aha (2015). Case-Based Behavior Recognition in Beyond Visual Range Air Combat. In: *Proceedings of the Twenty-Eighth International Florida Artificial Intelligence Research Society Conference (FLAIRS)*. Hollywood, Florida: AAAI Press.

Bou Ammar, H., S. Chen, K. Tuyls and G. Weiss (2014). Automated Transfer for Reinforcement Learning Tasks. In: *KI - Künstliche Intelligenz* 28.1, pp. 7–14. ISSN: 1610-1987. DOI: 10.1007/s13218-013-0286-8.

Bourassa, M. A. J. and L. Massey (2012). *Artificial Intelligence in Games. A Survey of the State of the Art*. Tech. rep. DRDC-OTTAWA-TM-2012-084. Defence R&D Canada - Ottawa, Ottawa ONT (CAN).

Bourassa, M., N. Abdellaoui and G. Parkinson (2011). Agent-Based Computer-Generated-Forces' Behaviour Improvement. In: *Proceedings of the 3rd International Conference on Agents and Artificial Intelligence*, pp. 273–280. ISBN: 978-989-8425-41-6. DOI: 10.5220/0003187002730280.

Bowling, M., N. Burch, M. Johanson and O. Tammelin (2015). Heads-up limit hold'em poker is solved. In: *Science* 347.6218, pp. 145–149. DOI: 10.1126/science.1259433.

Bruzzone, A. G. and M. Massei (2017). Simulation-Based Military Training. In: *Guide to Simulation-Based Disciplines: Advancing Our Computational Future*. Ed. by S. Mittal, U. Durak and T. Ören. Cham: Springer International Publishing, pp. 315–361. ISBN: 978-3-319-61264-5. DOI: 10.1007/978-3-319-61264-5_14.

Buşoniu, R. Babuška and B. de Schutter (2010). Multi-agent reinforcement learning: An overview. In: *Innovations in multi-agent systems and applications – 1*. Ed. by D. Srinivasan and L. C. Jain. Vol. 310. Studies in Computational Intelligence. Berlin, Germany: Springer. Chap. 7, pp. 183–221. DOI: 10.1007/978-3-642-14435-6_7.

Chao, X., G. Kou, T. Li and Y. Peng (2018). Jie Ke versus AlphaGo: A ranking approach using decision making method for large-scale data with incomplete information. In: *European Journal of Operational Research* 265.1, pp. 239–247. DOI: 10.1016/j.ejor.2017.07.030.

Chapman, R. and C. Colegrove (2013). Transforming operational training in the Combat Air Forces. In: *Military Psychology* 25.3, p. 177. DOI: 10.1037/h0095980.

Chen, S.-Y., Y. Yu, Q. Da, J. Tan, H.-K. Huang and H.-H. Tang (2018). Stabilizing Reinforcement Learning in Dynamic Environment with Application to Online Recommendation. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '18. London, United Kingdom: ACM, pp. 1187–1196. ISBN: 978-1-4503-5552-0. DOI: 10.1145/3219819.3220122.

Chrabaszcz, P., I. Loshchilov and F. Hutter (2018). Back to Basics: Benchmarking Canonical Evolution Strategies for Playing Atari. In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.* Pp. 1419–1426. DOI: 10.24963/ijcai.2018/197.

Church, A. M. (2015). The readiness crunch (combat units struggle to keep their full range of skills sharp). In: *Air Force Magazine* 98.3, pp. 40–43.

Coman, A. and H. Muñoz-Avila (2013). Automated Generation of Diverse NPC-controlling FSMs Using Nondeterministic Planning Techniques. In: *Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AIIDE'13. Boston, MA, USA: AAAI Press, pp. 121–127. ISBN: 978-1-57735-607-3.

Connors, C. D., J. Miller and B. J. Lunday (2016). Using agent-based modeling and a designed experiment to simulate and analyze a new air-to-air missile. In: *The Journal of Defense Modeling and Simulation* 13.3, pp. 321–330.

Crane, P., W. Bennett Jr, A. Borgvall and C. Waldelöf (2006). Advancing Fighter Employment Tactics in the Swedish and US Air Forces Using Simulation Environments. In: Meeting Proceedings RTO-MP-MSG-045. Neuilly-sur-Seine, France.

Dahlbom, A. and L. Niklasson (2006). Goal-directed hierarchical dynamic scripting for RTS games. In: *Proceedings of the Second Artificial Intelligence and Interactive Digital Entertainment Conference*, pp. 21–28.

Dal Pozzolo, A., O. Caelen, Y.-A. Le Borgne, S. Waterschoot and G. Bontempi (2014). Learned lessons in credit card fraud detection from a practitioner perspective. In: *Expert systems with applications* 41.10, pp. 4915–4928. DOI: `10.1016/j.eswa.2014.02.026`.

Darken, R. P. and C. L. Blais (2017). The Uniformed Military Modeling and Simulation Professional. In: *The Profession of Modeling and Simulation*. John Wiley & Sons, Inc. Chap. 8, pp. 151–166. ISBN: 9781119288091. DOI: `10.1002/9781119288091.ch8`.

Day, O. and T. M. Khoshgoftaar (2017). A survey on heterogeneous transfer learning. In: *Journal of Big Data* 4.1, p. 29. ISSN: 2196-1115. DOI: `10.1186/s40537-017-0089-0`.

De Winter, J. C. (2013). Using the Student's t-test with extremely small sample sizes. In: *Practical Assessment, Research & Evaluation* 18.10.

Debnath, S. C., B. B. Lee and S. Tandon (2015). Fifty Years and Going Strong: What Makes Behaviorally Anchored Rating Scales So Perennial as an Appraisal Method? In: *International Journal of Business and Social Science* 6.2.

Defense Advanced Research Projects Agency (DARPA) (2019). *Training AI to Win a Dogfight*. URL: `https://www.darpa.mil/news-events/2019-05-08` (visited on 27/10/2019).

DeNisi, A. S. and K. R. Murphy (2017). Performance appraisal and performance management: 100 years of progress? In: *Journal of Applied Psychology* 102.3, pp. 421–433. DOI: `10.1037/apl0000085`.

Derrick, B. and P. White (2017). Comparing two samples from an individual Likert question. In: *International Journal of Mathematics and Statistics* 18 (3), pp. 1–13.

Doyle, M. J., E. Watz and A. M. Portrey (2015). Merging Worlds: Complex Adaptive Systems Science Meets Systems Engineering: A Foundation for Complex Adaptive Agent-based Modeling Architectures. In: *Proceedings of the 48th Annual Simulation Symposium*. ANSS '15. Alexandria, Virginia: Society for Computer Simulation International, pp. 86–93. ISBN: 978-1-5108-0099-1.

Doyle, M. J. and A. M. Portrey (2014). Rapid adaptive realistic behavior modeling is viable for use in training. In: *Proceedings of the 23rd Conference on Behavior Representation in Modeling and Simulation (BRIMS)*, pp. 73–80.

Durak, U., O. Topçu, R. Siegfried and H. Oğuztüzün (2014). Scenario development: A model-driven engineering perspective. In: *Simulation and Modeling Methodologies, Technologies* *and Applications (SIMULTECH), 2014 International Conference on*. IEEE, pp. 117–124. DOI: 10.5220/0005009501170124.

Edwards, A. D., L. Downs and J. C. Davidson (2018). Forward-Backward Reinforcement Learning. In: *CoRR* abs/1803.10227.

EuroSim (2017). *EuroSim Real-Time Simulation Framework*. URL: http://www.eurosim.nl (visited on 10/11/2017).

Evertsz, R., J. Thangarajah and M. Papasimeon (2017). The Conceptual Modelling of Dynamic Teams for Autonomous Systems. In: *Conceptual Modeling: 36th International Conference, ER 2017, Valencia, Spain, November 6–9, 2017, Proceedings*. Ed. by H. C. Mayr, G. Guizzardi, H. Ma and O. Pastor. Cham: Springer International Publishing, pp. 311–324. ISBN: 978-3-319-69904-2. DOI: 10.1007/978-3-319-69904-2_25.

Ferrucci, D., E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager, N. Schlaefer and C. Welty (2010). Building Watson: An Overview of the DeepQA Project. In: *AI Magazine* 31.3, p. 59. DOI: 10.1609/aimag.v31i3.2303.

Fletcher, J. and A. P. Wind (2014). The evolving definition of cognitive readiness for military operations. In: *Teaching and measuring cognitive readiness*. Springer, pp. 25–52. DOI: 10.1007/978-1-4614-7579-8_2.

Floyd, M. W., J. Karneeb, P. Moore and D. W. Aha (2017). A Goal Reasoning Agent for Controlling UAVs in Beyond-Visual-Range Air Combat. In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization. DOI: 10.24963/ijcai.2017/657.

Foerster, J., Y. M. Assael, N. de Freitas and S. Whiteson (2016). Learning to communicate with deep multi-agent reinforcement learning. In: *Advances in Neural Information Processing Systems*, pp. 2137–2145.

Foster, R. E. and J. Fletcher (2013). Toward training transformation. In: *Military Psychology* 25.3, p. 308. DOI: 10.1037/h0094971.

Fu, D., R. Houlette and R. Jensen (2003). A visual environment for rapid behavior definition. In: *Proceedings of the 12th Conference on Behavior Representation in Modeling and Simulation (BRIMS)*.

Fürnkranz, J., T. Kliegr and H. Paulheim (2018). On cognitive preferences and the plausibility of rule-based models. In: *arXiv preprint arXiv:1803.01316*.

Gerretsen, A., J. van Oijen, G. R. Ferdinandus and P. G. M. Kerbusch (2017). *Towards more effective and efficient tactical scenario generation*. Tech. rep. TP-2017-175, TO APPEAR. Netherlands Aerospace Centre.

Goerger, S. R., M. L. McGinnis and R. P. Darken (2005). A validation methodology for human behavior representation models. In: *The Journal of Defense Modeling and Simulation* 2.1, pp. 39–51. DOI: 10.1177/154851290500200105.

Goldberg, B., F. Davis, J. M. Riley and M. W. Boyce (2017). Adaptive Training Across Simulations in Support of a Crawl-Walk-Run Model of Interaction. In: *Augmented Cognition. Enhancing*

Cognition and Behavior in Complex Human Environments: 11th International Conference, AC 2017, Held as Part of HCI International 2017, Vancouver, BC, Canada, July 9-14, 2017, Proceedings, Part II. Ed. by D. D. Schmorrow and C. M. Fidopiastis. Cham: Springer International Publishing, pp. 116–130. ISBN: 978-3-319-58625-0. DOI: 10.1007/978-3-319-58625-0_8.

Goyal, A., P. Brakel, W. Fedus, T. Lillicrap, S. Levine, H. Larochelle and Y. Bengio (2018). Recall Traces: Backtracking Models for Efficient Reinforcement Learning. In: *arXiv preprint arXiv:1804.00379v1*.

Grondman, I., L. Busoniu, G. A. D. Lopes and R. Babuska (2012). A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.6, pp. 1291–1307. ISSN: 1094-6977. DOI: 10.1109/TSMCC.2012.2218595.

Grześ, M. (2017). Reward shaping in episodic reinforcement learning. In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, pp. 565–573.

Hadfield-Menell, D., S. Milli, P. Abbeel, S. J. Russell and A. Dragan (2017). Inverse reward design. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett. Curran Associates, Inc., pp. 6765–6774.

Hahn, H. A. (2013). The conundrum of verification and validation of social science-based models. In: *Procedia Computer Science* 16, pp. 878–887.

— (2017). The Conundrum of Verification and Validation of Social Science-Based Models Redux. In: *Advances in Cross-Cultural Decision Making*. Springer, pp. 279–292.

Hameed, A., A. Khoshkbarforoushha, R. Ranjan, P. P. Jayaraman, J. Kolodziej, P. Balaji, S. Zeadally, Q. M. Malluhi, N. Tziritas, A. Vishnu et al. (2016). A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems. In: *Computing* 98.7, pp. 751–774. DOI: 10.1007/s00607-014-0407-8.

Hasselt, H. van (2010). Double Q-learning. In: *Advances in Neural Information Processing Systems*, pp. 2613–2621.

— (2011). Insights in Reinforcement Learning. PhD thesis. Utrecht University.

Havrylov, S. and I. Titov (2017). Emergence of Language with Multi-agent Games: Learning to Communicate with Sequences of Symbols. In: *arXiv preprint arXiv:1705.11192*.

Heidrich-Meisner, V., M. Lauer, C. Igel and M. A. Riedmiller (2007). Reinforcement learning in a nutshell. In: *ESANN 2007, 15th European Symposium on Artificial Neural Networks, Bruges, Belgium, April 25-27, 2007, Proceedings*, pp. 277–288.

Henninger, A. E., A. J. Gonzalez, M. Georgiopoulos and R. F. DeMara (2000). Modeling Semi-Automated Forces with Neural Networks: Performance Improvement through a Modular Approach. In: *Proceedings of the Ninth Conference on Computer Generated Forces and Behavioral Representation (CGF-BR'00)*. Orlando, FL.

Hinman, E., T. Jahn and J. Jinnette (2009). *AirLandBattle21: Transformational Concepts for Integrating Twenty-first Century Air and Ground Forces*. Ashgate. ISBN: 9780754676348.

Hoffman, R. R. (2014). *The psychology of expertise: Cognitive research and empirical AI*. Psychology Press. ISBN: 9780805819007. DOI: 10.4324/9781315806105.

Holmes, D., P. Moody, D. Dine and L. Trueman (2016). *Research Methods for the Biosciences*. Oxford University Press. ISBN: 9780198728498.

Hou, Y., Y.-S. Ong, L. Feng and J. M. Zurada (2017a). An Evolutionary Transfer Reinforcement Learning Framework for Multiagent Systems. In: *IEEE Transactions on Evolutionary Computation* 21.4, pp. 601–615. DOI: 10.1109/tevc.2017.2664665.

Hou, Y., F. Wei, S. X. Li, Z. Huang and A. Ashley (2017b). Coordination and performance analysis for a three-echelon supply chain with a revenue sharing contract. In: *International Journal of Production Research* 55.1, pp. 202–227. DOI: 10.1080/00207543.2016.1201601.

IEEE (2012). IEEE Standard for Distributed Interactive Simulation–Application Protocols. In: *IEEE Std 1278.1-2012 (Revision of IEEE Std 1278.1-1995)*, pp. 1–747. DOI: 10.1109/IEEESTD.2012.6387564.

Janssen, C. P. and W. D. Gray (2012). When, what, and how much to reward in reinforcement learning-based models of cognition. In: *Cognitive science* 36.2, pp. 333–358. DOI: 10.1111/j.1551-6709.2011.01222.x.

Jennings, N. R., K. Sycara and M. Wooldridge (1998). A Roadmap of Agent Research and Development. In: *Autonomous Agents and Multi-Agent Systems* 1.1, pp. 7–38. ISSN: 1387-2532. DOI: 10.1023/A:1010090405266.

Jordan, M. I. and T. M. Mitchell (2015). Machine learning: Trends, perspectives, and prospects. In: *Science* 349.6245, pp. 255–260. DOI: 10.1126/science.aaa8415.

Joseph, M., M. Kearns, J. H. Morgenstern and A. Roth (2016). Fairness in learning: Classic and contextual bandits. In: *Advances in Neural Information Processing Systems*, pp. 325–333.

Juzek, T. S. (2016). Acceptability judgement tasks and grammatical theory. PhD thesis. University of Oxford.

Kamrani, F., L. J. Luotsinen and R. A. Løvlid (2016). Learning objective agent behavior using a data-driven modeling approach. In: *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 2175–2181. DOI: 10.1109/SMC.2016.7844561.

Kaneshige, J. and K. Krishnakumar (2007). Artificial immune system approach for air combat maneuvering. In: *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*. Vol. 6560, p. 7. DOI: 10.1117/12.718892.

Kanetsuki, Y., R. Thawonmas and S. Nakata (2015). Optimization and simplification of dynamic scripting with evolution strategy and fuzzy control in a fighting game AI. In: *2015 IEEE 4th Global Conference on Consumer Electronics (GCCE)*. IEEE, pp. 330–331.

Kaufmann, E., O. Cappé and A. Garivier (2016). On the complexity of best-arm identification in multi-armed bandit models. In: *The Journal of Machine Learning Research* 17.1, pp. 1–42.

Kaushik, R., K. Chatzilygeroudis and J.-B. Mouret (2018). Multi-objective Model-based Policy Search for Data-efficient Learning with Sparse Rewards. In: *arXiv preprint arXiv:1806.09351v1*.

Khatami, A., P. Huibers and J. J. Roessingh (2013). 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 (BRiMS 2013)*. Ed. by B. Kennedy, R. Reitter and S. Amant. Ottawa, Canada: BRIMS Society.

Kim, J. H., S. Jeong, S. Oh and Y. J. Jang (2015). Verification, Validation, and Accreditation (VV&A) Considering Military and Defense Characteristics. In: *Industrial Engineering & Management Systems* 14.1, pp. 88–93. DOI: 10.7232/iems.2015.14.1.088.

Kitazato, Y. and S. Arai (2018). Estimation of Reward Function Maximizing Learning Efficiency in Inverse Reinforcement Learning. In: *Proceedings of the 10th International Conference on Agents and Artificial Intelligence, ICAART 2018, Volume 2, Funchal, Madeira, Portugal, January 16-18, 2018*. Pp. 276–283. DOI: 10.5220/0006729502760283.

Konokman, H. E., A. Kayran and M. Kaya (2017). Aircraft vulnerability assessment against fragmentation warhead. In: *Aerospace Science and Technology* 67, pp. 215–227. DOI: 10.1016/j.ast.2017.04.005.

Koo, T. K. and M. Y. Li (2016). A Guideline of Selecting and Reporting Intraclass Correlation Coefficients for Reliability Research. In: *Journal of Chiropractic Medicine* 15.2, p. 155. DOI: 10.1016/j.jcm.2016.02.012.

Koopmanschap, R., M. Hoogendoorn and J. J. Roessingh (2013). Learning Parameters for a Cognitive Model on Situation Awareness. In: *Recent Trends in Applied Artificial Intelligence: 26th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2013, Amsterdam, The Netherlands, June 17-21, 2013. Proceedings*. Ed. by M. Ali, T. Bosse, K. V. Hindriks, M. Hoogendoorn, C. M. Jonker and J. Treur. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 22–32. ISBN: 978-3-642-38577-3. DOI: 10.1007/978-3-642-38577-3_3.

— (2015). Tailoring a cognitive model for situation awareness using machine learning. In: *Applied Intelligence* 42.1, pp. 36–48. ISSN: 0924-669X. DOI: 10.1007/s10489-014-0584-3.

Lakens, D. (2017). Equivalence Tests: A Practical Primer for t Tests, Correlations, and Meta-Analyses. In: *Social Psychological and Personality Science* 8.4, pp. 355–362. ISSN: 1948-5514. DOI: 10.1177/1948550617697177.

Laslie, B. (2015). *The Air Force Way of War: U.S. Tactics and Training After Vietnam*. University Press of Kentucky. ISBN: 9780813160863.

Lazaric, A. (2012). Transfer in Reinforcement Learning: A Framework and a Survey. English. In: *Reinforcement Learning*. Ed. by M. Wiering and M. van Otterlo. Vol. 12. Adaptation, Learning, and Optimization. Springer Berlin Heidelberg, pp. 143–173. ISBN: 978-3-642-27644-6. DOI: 10.1007/978-3-642-27645-3_5.

Lee, D., H. Tang, J. O. Zhang, H. Xu, T. Darrell and P. Abbeel (2018). Modular Architecture for StarCraft II with Deep Reinforcement Learning. In: *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Leuenberger, G. and M. A. Wiering (2018). Actor-Critic Reinforcement Learning with Neural Networks in Continuous Games. In: *Proceedings of the 10th International Conference on Agents and Artificial Intelligence*. Scitepress - Science and Technology Publications. DOI: 10.5220/0006556500530060.

Liu, P. and Y. Ma (2017). A Deep Reinforcement Learning Based Intelligent Decision Method for UCAV Air Combat. In: *Modeling, Design and Simulation of Systems: 17th Asia Simulation Conference, AsiaSim 2017, Melaka, Malaysia, August 27 – 29, 2017, Proceedings, Part I*. Ed. by M. S. Mohamed Ali, H. Wahid, N. A. Mohd Subha, S. Sahlan, M. A. Md. Yunus and A. R. Wahap. Singapore: Springer Singapore, pp. 274–286. ISBN: 978-981-10-6463-0. DOI: 10.1007/978-981-10-6463-0_24.

Lopes, R. and R. Bidarra (2011). Adaptivity challenges in games and simulations: a survey. In: *Computational Intelligence and AI in Games, IEEE Transactions on* 3.2, pp. 85–99.

Løvlid, R. A., A. Alstad, O. M. Mevassvik, N. de Reus, H. Henderson, B. van der Vecht and T. Luik (2013). Two approaches to developing a multi-agent system for battle command simulation. In: *Proceedings of the 2013 Winter Simulation Conference: Simulation: Making Decisions in a Complex World*. IEEE Press, pp. 1491–1502. DOI: 10.1109/wsc.2013.6721533.

Lu, B. and G. Gong (2014). A method for computer generated forces behavior modeling based on composability. In: *Guidance, Navigation and Control Conference (CGNCC), 2014 IEEE Chinese*. IEEE, pp. 2038–2041. DOI: 10.1109/CGNCC.2014.7007490.

Lu, J., V. Behbood, P. Hao, H. Zuo, S. Xue and G. Zhang (2015). Transfer learning using computational intelligence: A survey. In: *Knowledge-Based Systems* 80, pp. 14–23. DOI: 10.1016/j.knosys.2015.01.010.

Ludwig, J. R. and A. Farley (2008). Using hierarchical dynamic scripting to create adaptive adversaries. In: *Proceedings of the 17th Conference on Behavior Representation in Modeling and Simulation (BRIMS)*.

Luotsinen, L. J., F. Kamrani, P. Hammar, M. Jändel and R. A. Løvlid (2016). Evolved creative intelligence for computer generated forces. In: *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 3063–3070. DOI: 10.1109/SMC.2016.7844707.

Ma, Y., X. Ma and X. Song (2014). A case study on air combat decision using approximated dynamic programming. In: *Mathematical Problems in Engineering* 2014.

MacLeod, M. R. (2012). Preventing Premature Conclusions: Analysis of Human-In-the-Loop Air Combat Simulations. In: *The 29th International Symposium on Military Operational Research*.

MacMillan, J., E. B. Entin, R. Morley and W. Bennett Jr (2013). Measuring team performance in complex and dynamic military environments: The SPOTLITE method. In: *Military Psychology* 25.3, p. 266.

Majchrzak, K., J. Quadflieg and G. Rudolph (2015). Advanced Dynamic Scripting for FightingGameAI. English. In: *Entertainment Computing - ICEC 2015*. Ed. by K. Chorianopoulos, M. Divitini, J. Baalsrud Hauge, L. Jaccheri and R. Malaka. Vol. 9353. Lecture Notes in Computer Science. Springer International Publishing, pp. 86–99. ISBN: 978-3-319-24588-1. DOI: 10.1007/978-3-319-24589-8_7.

Marcus, S. (2013). *Automating knowledge acquisition for expert systems*. Vol. 57. Springer Science & Business Media. ISBN: 978-1-4684-7122-9.

Marken, R., W. Taylor, J. Ausink, L. Hanser and C. Anderegg (2007). *Absorbing and Developing Qualified Fighter Pilots: The Role of the Advanced Simulator*. RAND Corporation. ISBN: 9780833044457.

Marzinotto, A., M. Colledanchise, C. Smith and P. Ögren (2014). Towards a unified behavior trees framework for robot control. In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, pp. 5420–5427. DOI: 10.1109/icra.2014.6907656.

Mattingly, C., A. Bolton, M. Walwanis and H. W. Priest (2014). Game on: Live, virtual, constructive training can improve readiness. In: *Future Force* 1.3, pp. 34–37.

McLean, G. M. T., S. Lambeth and T. Mavin (2016). The Use of Simulation in Ab Initio Pilot Training. In: *The International Journal of Aviation Psychology* 26.1-2, pp. 36–45. DOI: 10.1080/10508414.2016.1235364.

McLennan, B., J. Molloy, J. Whittaker and J. Handmer (2016). Centralised coordination of spontaneous emergency volunteers: the EV CREW model. In: *Australian Journal of Emergency Management, The* 31.1, p. 24.

McMahon, D. C. (1990). A neural network trained to select aircraft maneuvers during air combat: a comparison of network and rule based performance. In: *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, 107–112 vol.1. DOI: 10.1109/IJCNN.1990.137554.

Merk, R.-J. (2013). Making Enemies: Cognitive Modelling for Opponent Agents in Fighter Pilot Simulators. PhD thesis. VU University Amsterdam, p. 231.

Meyners, M. (2012). Equivalence tests – A review. In: *Food Quality and Preference* 26.2, pp. 231–245. ISSN: 0950-3293. DOI: 10.1016/j.foodqual.2012.05.003.

Mills, C. (2009). *Breaking the Kill Chain*. Tech. rep. APA-NOTAM-270109-1. Air Power Australia.

Mittal, S., M. J. Doyle and E. Watz (2013). Detecting intelligent agent behavior with environment abstraction in complex air combat systems. In: *2013 IEEE International Systems Conference (SysCon)*, pp. 662–670. DOI: 10.1109/SysCon.2013.6549953.

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski et al. (2015). Human-level control through deep reinforcement learning. In: *Nature* 518.7540, pp. 529–533.

Mulgund, S., K. Harper and K. Krishnakumar (1998). Air Combat Tactics Optimization using Stochastic Genetic Algorithms. In: *Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on*. Vol. 4. IEEE, pp. 3136–3141. DOI: 10.1109/ICSMC.1998.726484.

Mulgund, S., K. Harper and G. Zacharias (2001). Large-scale air combat tactics optimization using genetic algorithms. In: *Journal of Guidance, Control, and Dynamics* 24.1, pp. 140–142. DOI: 10.2514/2.4689.

Naval Air Systems Command (2010). *Personal Computer Debriefing System*. URL: http://www.navair.navy.mil/tande/ranges/ATR/docs/atr/FS_PCDS.pdf (visited on 10/11/2017).

Netherlands Aerospace Centre (2017a). *Smart Bandits AIR - NLR*. URL: http://www.nlr.org/capabilities/smart-bandits-air/ (visited on 12/07/2017).

— (2017b). *Fighter 4-Ship research simulation facility*. URL: http://www.nlr.nl/downloads/f281.pdf (visited on 06/11/2017).

Norman, G. (2010). Likert scales, levels of measurement and the "laws" of statistics. In: *Advances in health sciences education* 15.5, pp. 625–632.

Nowé, A. and T. Brys (2016). A Gentle Introduction to Reinforcement Learning. In: *Scalable Uncertainty Management*. DOI: 10.1007/978-3-319-45856-4_2.

Olde, B. A. and J. DiCola (2014). How can we make computer-generated forces more real? In: *Future Force* 1.3, pp. 34–37.

Ömer, F. A. and K. Ayan (2013). A flexible rule-based framework for pilot performance analysis in air combat simulation systems. In: *Turkish Journal of Electrical Engineering & Computer Sciences* 21.Sup. 2, pp. 2397–2415.

Ortega, J., N. Shaker, J. Togelius and G. N. Yannakakis (2013). Imitating human playing styles in super mario bros. In: *Entertainment Computing* 4.2, pp. 93–104.

Ososky, S., R. Sottilare, K. Brawner, R. Long and A. Graesser (2015). *Authoring Tools and Methods for Adaptive Training and Education in Support of the US Army Learning Model: Research Outline*. Tech. rep. ARL-SR-0339. US Army Research Laboratory Aberdeen Proving Ground, United States.

Oswalt, I. and T. Cooley (2019). Simulation Based Training's Incorporation of Machine Learning. In: *Proceedings of MODSIM World*.

Pan, S. J. and Q. Yang (2010). A survey on transfer learning. In: *Knowledge and Data Engineering, IEEE Transactions on* 22.10, pp. 1345–1359. DOI: 10.1109/tkde.2009.191.

Panait, L. and S. Luke (2005). Cooperative Multi-Agent Learning: The State of the Art. In: *Autonomous Agents and Multi-Agent Systems* 11.3, pp. 387–434. ISSN: 1387-2532. DOI: 10.1007/s10458-005-2631-2.

Paparone, C. (2017). How we fight: A critical exploration of US military doctrine. In: *Organization* 24.4, pp. 516–533. DOI: 10.1177/1350508417693853.

Pelosi, M. J. and M. S. Brown (2016). Software engineering a multi-layer and scalable autonomous forces AI for professional military training. In: *Proceedings of the 2016 Winter Simulation Conference*. IEEE Press, pp. 3122–3133. DOI: 10.1109/WSC.2016.7822345.

Peña-Ayala, A. (2014). Educational data mining: A survey and a data mining-based analysis of recent works. In: *Expert systems with applications* 41.4, pp. 1432–1462. DOI: 10.1016/j.eswa.2013.08.042.

Petrik, M. and B. Scherrer (2009). Biasing approximate dynamic programming with a lower discount factor. In: *Advances in neural information processing systems*, pp. 1265–1272.

Petty, M. D. (2003). Benefits and Consequences of Automated Learning in Computer Generated Forces Systems. In: *INFORMATION AND SECURITY* 12, pp. 63–74. DOI: 10.11610/isij.1203.

— (2010). Verification, validation, and accreditation. In: *Modeling and Simulation Fundamentals: Theoretical Underpinnings and Practical Domains*. Ed. by J. A. Sokolowski and C. M. Banks. John Wiley & Sons Hoboken, NJ, USA. Chap. 10, pp. 325–372. ISBN: 978-0-470-48674-0.

Petty, M. D. and S. E. Barbosa (2016). Improving Air Combat Maneuvering Skills Through Self-Study and Simulation-Based Practice. In: *Simulation & Gaming* 47.1, pp. 103–129. DOI: 10.1177/1046878116628236.

Phillips, J. K., J. Shafer, K. G. Ross, D. A. Cox and S. B. Shadrick (2006). *Behaviorally anchored rating scales for the assessment of tactical thinking mental models*. Tech. rep. 1854. DTIC Document.

Policarpo, D., P. Urbano and T. Loureiro (2010). Dynamic scripting applied to a First-Person Shooter. In: *Information Systems and Technologies (CISTI), 2010 5th Iberian Conference on*. IEEE, pp. 1–6.

Ponsen, M., H. Muñoz-Avila, P. Spronck and D. W. Aha (2005). Automatically acquiring domain knowledge for adaptive game AI using evolutionary learning. In: *Artificial Intelligence* 20.3, pp. 1535–1540.

Ponsen, M., P. Spronck, H. Muñoz-Avila and D. W. Aha (2007). Knowledge acquisition for adaptive game AI. In: *Science of Computer Programming* 67.1, pp. 59–75. ISSN: 01676423. DOI: 10.1016/j.scico.2007.01.006.

Popov, I., N. Heess, T. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez and M. Riedmiller (2017). Data-efficient Deep Reinforcement Learning for Dexterous Manipulation. In: *arXiv preprint arXiv:1704.03073v1*.

Presagis (2012). *STAGE | Scenario Generation Software*. URL: http://www.presagis.com/products_services/products/modeling-simulation/simulation/stage/ (visited on 10/11/2017).

Ratner, E., D. Hadfield-Menell and A. Dragan (2018). Simplifying Reward Design through Divide-and-Conquer. In: *Robotics: Science and Systems XIV*. Robotics: Science and Systems Foundation. DOI: 10.15607/rss.2018.xiv.048.

Rodin, E. Y. and S. M. Amin (1992). Maneuver prediction in air combat via artificial neural networks. In: *Computers & mathematics with applications* 24.3, pp. 95–112. DOI: 10.1016/0898-1221(92)90217-6.

Rodriguez-Aguilar, J. A., C. Sierra, J. L. Arcos, M. López-Sánchez and I. Rodriguez (2015). Towards next generation coordination infrastructures. In: *The Knowledge Engineering Review* 30.4, p. 435.

Roessingh, J. J. M., R. Rijken, R. Merk, R. T. A. Meiland, P. F. Huibers, T. K. Lue and C. Montijn (2011). *Modelling CGFs for tactical air-to-air combat training: Motivation-based behaviour and Machine Learning in a common architecture*. Tech. rep. TP-2011-540. Amsterdam, the Netherlands: National Aerospace Laboratory NLR, p. 18.

Sadagic, A. (2010). Validating Visual Simulation of Small Unit Behavior. In: *Proceedings of the 2010 Interservice/Industry Training, Simulation, and Education Conference*. Orlando, Florida: I/ITSEC.

Santoso, S. and I. Supriana (2014). Minimax guided reinforcement learning for turn-based strategy games. In: *Information and Communication Technology (ICoICT), 2014 2nd International Conference on*. IEEE, pp. 217–220.

Sargent, E. (1939). The Link Trainer. In: *Royal United Services Institution. Journal* 84.535, pp. 590–592. DOI: 10.1080/03071843909419931.

Sargent, R. G. (2011). Verification and Validation of Simulation Models. In: *Proceedings of the Winter Simulation Conference*. WSC '11. Phoenix, Arizona: Winter Simulation Conference, pp. 183–198.

Schreiber, B. T., M. Schroeder and W. Bennett Jr. (2011). Distributed Mission Operations Within-Simulator Training Effectiveness. In: *The International Journal of Aviation Psychology* 21.3, pp. 254–268. DOI: 10.1080/10508414.2011.582448.

Schuirmann, D. J. (1987). A comparison of the two one-sided tests procedure and the power approach for assessing the equivalence of average bioavailability. In: *Journal of Pharmacokinetics and Pharmacodynamics* 15.6, pp. 657–680. DOI: 10.1007/BF01068419.

Shaffer, D. W., A. Ruis and A. C. Graesser (2015). Authoring networked learner models in complex domains. In: *Design Recommendations for Intelligent Tutoring Systems: Authoring Tools and Expert Modeling Techniques*. Ed. by R. A. Sottilare, A. C. Graesser and K. Hu Xiangen & Brawner. Vol. 3. Adaptive Tutoring. Chap. 13. ISBN: 9780989392372.

Shao, L., F. Zhu and X. Li (2015). Transfer Learning for Visual Categorization: A Survey. In: *IEEE Transactions on Neural Networks and Learning Systems* 26.5, pp. 1019–1034. ISSN: 2162-237X. DOI: 10.1109/TNNLS.2014.2330900.

Shaw, R. L. (1985). *Fighter Combat: Tactics and Maneuvering*. 6th edition. Naval Institute Press, p. 428. ISBN: 978-0870210594.

Silva, M. P., V. do Nascimento Silva and L. Chaimowicz (2015). Dynamic difficulty adjustment through an adaptive AI. In: *Computer Games and Digital Entertainment (SBGames), 2015 14th Brazilian Symposium on*. IEEE, pp. 173–182. DOI: 10.1109/SBGames.2015.16.

Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot et al. (2016). Mastering the game of Go with deep neural networks and tree search. In: *Nature* 529.7587, pp. 484–489. DOI: 10.1038/nature16961.

Silver, D., T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan and D. Hassabis (2017a). Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. In: *arXiv preprint arXiv:1712.01815*.

Silver, D., J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel and D. Hassabis (2017b). Mastering the game of Go without human knowledge. In: *Nature* 550.7676, pp. 354–359. ISSN: 0028-0836. DOI: 10.1038/nature24270.

Smith, R., B. Dike, R. Mehra, B. Ravichandran and A. El-Fallah (2000a). Classifier systems in combat: two-sided learning of maneuvers for advanced fighter aircraft. In: *Computer Methods in Applied Mechanics and Engineering* 186.2-4, pp. 421–437. ISSN: 0045-7825. DOI: 10.1016/S0045-7825(99)00395-3.

Smith, R. E., B. A. Dike, B. Ravichandran, A. El-Fallah and R. K. Mehra (2000b). The fighter aircraft LCS: A case of different LCS goals and techniques. In: *Learning Classifier Systems*. Ed. by P. L. Lanzi, W. Stolzmann and S. W. Wilson. Springer Berlin Heidelberg, pp. 283–300. ISBN: 978-3-540-45027-6. DOI: 10.1007/3-540-45027-0_15.

Smith, R. (2010). The long history of gaming in military training. In: *Simulation & Gaming* 41.1, pp. 6–19. DOI: 10.1177/1046878109334330.

Snell, S., S. Morris and G. Bohlander (2015). *Managing Human Resources*. Cengage Learning. ISBN: 9781305480735.

Sottilare, R. (2013). Training Technology, the State of Practice and Emerging Concepts. In: *Fundamental Issues in Defense Training and Simulation*. Ed. by C. Best, G. Galanis, J. Kerry and R. Sottilare. Brookfield, VT, USA: Ashgate Publishing Company. ISBN: 9781409447214.

Spector, B. and S. Belongie (2018). Sample-Efficient Reinforcement Learning through Transfer and Architectural Priors. In: *arXiv preprint arXiv:1801.02268*.

Spronck, P., M. Ponsen, I. Sprinkhuizen-Kuyper and E. Postma (2006). Adaptive game AI with dynamic scripting. In: *Machine Learning* 63.3, pp. 217–248. ISSN: 08856125. DOI: 10.1007/s10994-006-6205-6.

Stacy, W. and J. Freeman (2016). Training objective packages: enhancing the effectiveness of experiential training. In: *Theoretical Issues in Ergonomics Science* 17.2, pp. 149–168. DOI: 10.1080/1463922X.2015.1111459.

Stillion, J. (2015). *Trends in Air-to-Air Combat: Implications for Future Air Superiority*. Tech. rep. Center for Strategic and Budgetary Assessments.

Stone, P. and M. Veloso (2000). Multiagent systems: A survey from a machine learning perspective. In: *Autonomous Robots* 8.3, pp. 345–383.

Stytz, M. R. and S. B. Banks (2003a). Progress and Prospects for the Development of Computer-Generated Actors for Military Simulation: Part 1-Introduction and Background. In: *Presence* 12.3, pp. 311–325. ISSN: 1054-7460. DOI: 10.1162/105474603765879558.

— (2003b). Progress and Prospects for the Development of Computer Generated Actors for Military Simulation, Part 3-The Road Ahead. In: *Presence* 12.6, pp. 629–643. ISSN: 1054-7460. DOI: 10.1162/105474603322955923.

Su, X., M. Zhang and Q. Bai (2016). Coordination for dynamic weighted task allocation in disaster environments with time, space and communication constraints. In: *Journal of Parallel and Distributed Computing* 97.Supplement C, pp. 47–56. ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2016.06.010.

Sutton, R. S. and A. G. Barto (1998). *Reinforcement Learning: An Introduction*. Vol. 1. 1. Cambridge Univ Press.

— (2018). *Reinforcement Learning: An Introduction*. Ed. by F. Bach. 2nd ed. Vol. 1. 1. Cambridge Univ Press.

Tatsumi, T., T. Komine, H. Sato and K. Takadama (2015). Handling different level of unstable reward environment through an estimation of reward distribution in XCS. In: *Proc. IEEE Congress Evolutionary Computation (CEC)*, pp. 2973–2980. DOI: 10.1109/CEC.2015.7257259.

Taylor, K. and D. Rohrer (2010). The effects of interleaved practice. In: *Applied Cognitive Psychology* 24.6, pp. 837–848. DOI: 10.1002/acp.1598.

Taylor, M. E. and P. Stone (2009). Transfer learning for reinforcement learning domains: A survey. In: *The Journal of Machine Learning Research* 10, pp. 1633–1685.

Teng, T.-H., A.-H. Tan, W.-S. Ong and K.-L. Lee (2012). Adaptive CGF for pilots training in air combat simulation. In: *15th International Conference on Information Fusion (FUSION)*. Singapore, pp. 2263–2270.

Teng, T.-H., A.-H. Tan and L.-N. Teow (2013). Adaptive computer-generated forces for simulator-based training. In: *Expert Systems with Applications* 40.18, pp. 7341–7353. DOI: 10.1016/j.eswa.2013.07.004.

Tetreault, J. R., D. Bohus and D. J. Litman (2007). Estimating the Reliability of MDP Policies: A Confidence Interval Approach. In: *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 276–283.

Thawonmas, R. and S. Osaka (2006). A method for online adaptation of computer-game AI rulebase. In: *International Conference on Advances in Computer Entertainment Technology 2006*. Screen Digest. ACM Press, p. 16. ISBN: 1595933808. DOI: 10.1145/1178823.1178843.

Timuri, T., P. Spronck and H. J. van den Herik (2007). Automatic rule ordering for dynamic scripting. In: *Proceedings of the Third Artificial Intelligence and Interactive Digital Entertainment Conference*.

Toubman, A. (2019). Validating Air Combat Behaviour Models for Adaptive Training of Teams. In: *Adaptive Instructional Systems*. Ed. by R. A. Sottilare and J. Schwarz. Springer International Publishing, pp. 557–571. DOI: 10.1007/978-3-030-22341-0_44.

Toubman, A., J. J. Roessingh, J. van Oijen, M. Hou, L. Luotsinen, J. Harris, R. A. Løvlid, C. Meyer, R. Rijken and M. Turčaník (2016a). Modeling Behavior of Computer Generated Forces with Machine Learning Techniques, the NATO task group approach. In: *Systems, Man, and Cybernetics (SMC), 2016 IEEE International Conference on*. Budapest, Hungary: IEEE. DOI: 10.1109/SMC.2016.7844517.

Toubman, A., J. J. Roessingh, P. Spronck, A. Plaat and H. J. van den Herik (2014a). Dynamic Scripting with Team Coordination in Air Combat Simulation. In: *Modern Advances in Applied Intelligence: 27th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2014, Kaohsiung, Taiwan, June 3-6, 2014, Proceedings, Part I*. Ed. by M. Ali, J.-S. Pan, S.-M. Chen and M.-F. Horng. Vol. 8481. Lecture Notes in Computer Science. Kaohsiung, Taiwan: Springer International Publishing, pp. 440–449. ISBN: 978-3-319-07455-9. DOI: 10.1007/978-3-319-07455-9_46.

— (2014b). Centralized Versus Decentralized Team Coordination Using Dynamic Scripting. In: *Proceedings of the 28th European Simulation and Modelling Conference - ESM'2014*. Ed. by A. C. Brito, J. M. R. Tavares and C. Braganca de Oliveira. Porto, Portugal: Eurosis, pp. 129–134.

— (2015a). Rewarding Air Combat Behavior in Training Simulations. In: *Systems, Man, and Cybernetics (SMC), 2015 IEEE International Conference on*. Hong Kong: IEEE Press, pp. 1397–1402. DOI: 10.1109/SMC.2015.248.

— (2015b). Transfer Learning of Air Combat Behavior. In: *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. Miami, Florida: IEEE Press, pp. 226–231. DOI: 10.1109/ICMLA.2015.61.

— (2016b). Rapid Adaptation of Air Combat Behaviour. In: *ECAI 2016 - 22nd European Conference on Artificial Intelligence*. Ed. by G. A. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum and F. van Harmelen. Vol. 285. Frontiers in Artificial Intelligence and Applications. The Hague, The Netherlands: IOS Press, pp. 1791–1796. DOI: 10.3233/978-1-61499-672-9-1791.

Tsifetakis, E. and T. Kontogiannis (2017). Evaluating non-technical skills and mission essential competencies of pilots in military aviation environments. In: *Ergonomics* 0.0. PMID: 28534423, pp. 1–15. DOI: 10.1080/00140139.2017.1332393.

Tuyls, K. and G. Weiss (2012). Multiagent learning: Basics, challenges, and prospects. In: *AI Magazine* 33.3, p. 41. DOI: 10.1609/aimag.v33i3.2426.

Tzu, S. (1994). *The art of war*. Ed. by D. C. Stevenson. Trans. by L. Giles. URL: http://classics.mit.edu/Tzu/artwar.html (visited on 15/09/2017).

US Department of Defense (2009). *DoD Modeling and Simulation (M&S) Verification, Validation, and Accreditation (VV&A). Department of Defense Instruction 5000.61.* URL: http://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodi/500061p.pdf.

Vakil, E. and E. Heled (2016). The effect of constant versus varied training on transfer in a cognitive skill learning task: The case of the Tower of Hanoi Puzzle. In: *Learning and Individual Differences* 47, pp. 207–214. DOI: 10.1016/j.lindif.2016.02.009.

Večerík, M., T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe and M. Riedmiller (2017). Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards. In: *arXiv preprint arXiv:1707.08817v1*.

Wang, S. L., K. Shafi, T. F. Ng, C. Lokan and H. A. Abbass (2017). Contrasting Human and Computational Intelligence Based Autonomous Behaviors in a Blue–Red Simulation Environment. In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 1.1, pp. 27–40. DOI: 10.1109/TETCI.2016.2641929.

Watkins, C. J. and P. Dayan (1992). Q-learning. In: *Machine learning* 8.3-4, pp. 279–292.

Wilcke, X., M. Hoogendoorn and J. J. Roessingh (2014). Co-evolutionary Learning for Cognitive Computer Generated Entities. In: *Modern Advances in Applied Intelligence: 27th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2014, Kaohsiung, Taiwan, June 3-6, 2014, Proceedings, Part II*. Ed. by M. Ali, J.-S. Pan, S.-M. Chen and M.-F. Horng. Cham: Springer International Publishing, pp. 120–129. ISBN: 978-3-319-07467-2. DOI: 10.1007/978-3-319-07467-2_13.

Wray, R. E., A. Woods, J. Haley and J. T. Folsom-Kovarik (2017). Evaluating Instructor Configurability for Adaptive Training. In: *Advances in Cross-Cultural Decision Making: Proceedings of the AHFE 2016 International Conference on Cross-Cultural Decision Making (CCDM), July 27-31,2016, Walt Disney World, Florida, USA*. Ed. by S. Schatz and M. Hoffman. Cham: Springer International Publishing, pp. 195–206. ISBN: 978-3-319-41636-6. DOI: 10.1007/978-3-319-41636-6_16.

Ximeng, X. U., Y. Rennong and F. U. Ying (2018). Situation assessment for air combat based on novel semi-supervised naive Bayes. In: *Journal of Systems Engineering and Electronics* 29.4, pp. 768–779.

Yan, Z., N. Jouandeau and A. A. Cherif (2013). A Survey and Analysis of Multi-Robot Coordination. In: *International Journal of Advanced Robotic Systems* 10.12, p. 399. DOI: 10.5772/57313.

Yannakakis, G. and J. Togelius (2014). A Panorama of Artificial and Computational Intelligence in Games. In: *Computational Intelligence and AI in Games, IEEE Transactions on* PP.99, pp. 1–1. ISSN: 1943-068X. DOI: 10.1109/TCIAIG.2014.2339221.

Yao, J., Q. Huang and W. Wang (2015). Adaptive Human Behavior Modeling for Air Combat Simulation. In: *IEEE/ACM 19th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pp. 100–103. DOI: 10.1109/DS-RT.2015.12.

Ye, D., M. Zhang and A. V. Vasilakos (2017). A survey of self-organization mechanisms in multiagent systems. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 47.3, pp. 441–461. DOI: 10.1109/TSMC.2015.2504350.

Yildiz, A., U. Akcal, B. Hostas, N. K. Ure and G. Inalhan (2018). Finite State Automata Based Approach to Autonomous Stall and Upset Recovery for Agile Aircraft. In: *2018 AIAA Guidance,* *Navigation, and Control Conference*. American Institute of Aeronautics and Astronautics. DOI: 10.2514/6.2018-1867.

Zhang, Q., L. Sun, P. Jiao and Q. Yin (2017). Combining behavior trees with MAXQ learning to facilitate CGFs behavior modeling. In: *2017 4th International Conference on Systems and Informatics (ICSAI)*, pp. 525–531. DOI: 10.1109/ICSAI.2017.8248348.

Zheng, W. and L. Feiguo (2017). Terminal efficiency of fragment air-to-air missile using Monte Carlo method. In: *Proc. 8th Int. Conf. Mechanical and Aerospace Engineering (ICMAE)*, pp. 730–735. DOI: 10.1109/ICMAE.2017.8038740.

Appendices

Appendix A

The Lightweight Air Combat Simulator

In this appendix we present the Lightweight Air Combat Simulator (LWACS). The appendix is organised as follows. First, we provide a general description of LWACS (Appendix A.1). Next, we describe the CGFS in LWACS (Appendix A.2), and then briefly introduce the scripting language by which behaviour models for the CGFS are created (Appendix A.3). Furthermore, we present the air combat scenarios that we have developed for use in LWACS (Appendix A.4).

A.1 Description

LWACS simulates a section of airspace. The simulated section of airspace is inhabited by air combat CGFS (see Appendix A.2) that engage each other in predefined scenarios (see Appendix A.4). LWACS was designed to require few computational resources to run, so that it can comfortably run many simulations in parallel on a modern desktop computer. The term "lightweight" in the name of the simulator refers to the low system requirements of the simulator.

The LWACS program is written in the Java programming language. It can be run in two modes: (1) with a graphical user interface (GUI) that allows inspection of the simulated airspace, and (2) with a command-line interface (CLI) (a.k.a. headless). In the headless mode, LWACS is able to run automated simulations in two ways: both (1) in faster-than-real-time and (2) in parallel. LWACS currently does not support human-in-the-loop simulations.

LWACS was developed at NLR in the context of the Smart Bandits (SB) project (see Appendix D). In the SB project, LWACS served as a platform for (1) testing behaviour models and (2) to explore the use of machine learning within air combat simulations. Because LWACS was completely developed at NLR, we had access to the entire source code. This allowed us to adapt LWACS to our research purposes (i.e., the automated simulations in Chapters 3, 4, and 5) as needed.

A.2 Computer generated forces

LWACS supports one kind of CGF, which represents a generic fighter jet. In LWACS, each CGF carries three types of devices: (1) a radar, (2) air-to-air missiles, and (3) a radar warning receiver. We describe the devices below.

Radar. The radar is a sensor that detects other aircraft. In the simulation, it produces a forward-looking cone, emanating from the front of the aircraft carrying the radar. Any aircraft inside this cone are detected by the radar. The radar has two modes: (1) the search mode, which produces a wide cone (120°), and (2) the tracking mode, which produces a narrow cone (30°). The search mode is used to find opposing aircraft. The tracking mode is used to track the movement of one specific opposing aircraft. The radar has a detect range of 100 km. Because of the limited range of the radar, deliberate manoeuvring is required to find and track opponents.

Air-to-air missiles. Each aircraft carries four air-to-air missiles. These missiles can only be fired at opponents. A prerequisite for firing a missile at an opponent is that the opponent should be tracked by the radar of the aircraft firing the missile. Hitting an opponent with a missile disables their aircraft and removes it from the simulation. Each aircraft only carries four missiles. Furthermore, missiles do not hit in a deterministic manner. Upon impact, each missile calculates its so-called probability-of-kill (P_k, see Chapter 4). The P_k of a missile decreases as the distance flown towards the target of the missile increases. The precise decrease of the P_k is defined by means of a predefined curve. The curve used in LWACS is shown in Figure 4.2. The P_k of missiles in LWACS starts at 1, and stays 1 until the missile has flown 50 km. From 50 km onward, the P_k of missiles declines until it reaches 0 after 80 km. After having flown 80 km, missiles are removed from the simulation.

Radar warning receiver (RWR). The RWR is a device that detects whether the aircraft is inside the radar cone of another aircraft. The pilot may use this information to assume that a missile will be fired or has already been fired at him, and then take action accordingly. The RWR has a detection range of 200 km.

A screenshot of LWACS is shown in Figure A.1. The CGFs in LWACS are graphically displayed as F-16 fighter jets. All CGFs belong to one of two teams: either (1) the blue team or (2) the red team. The graphical models of the CGFs are coloured to indicate team membership.

Three different indicators can be shown next to each CGF. The indicators appear as small coloured triangles. They serve to visualise the status of each CGF. The three indicators are as follows. First, a light blue indicator shows whether the CGF has detected another CGF by means of its radar. Second, a red indicator shows whether the CGF has detected another CGF by means of its RWR. Third, white indicators show the number of missiles remaining in the inventory of each CGF.

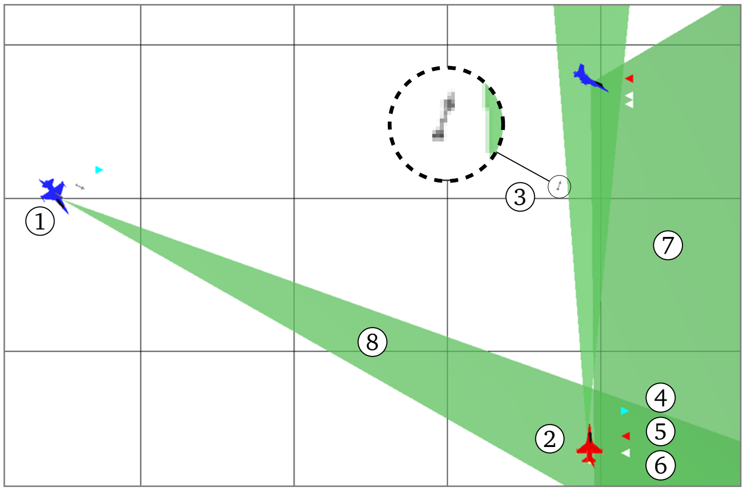


Figure A.1 A screenshot of LWACS, showing: (1) a blue CGF, (2) a red CGF, (3) a missile (with magnification), (4) a light blue radar indicator, (5) a red radar warning receiver indicator, (6) white missile inventory indicators, (7) a wide radar cone (search mode), (8) a narrow radar cone (tracking mode).

In LWACS, the CGFs have two restrictions on their movement. The first restriction is the use of a basic flight model. The flight model allows the CGFs to either (1) move at a constant velocity, (2) accelerate, and (3) decelerate, all as if the CGFs were in a vacuum. In other words, the flight model has no notion of aerodynamics or gravity. The second restriction is that the CGFs are only allowed to move in the horizontal plane. The reason for the second restriction is that the radars of the CGFs were unable to operate in a three-dimensional environment at the time of our research. Regardless, vertical movement in LWACS is meaningless because the flight model provides no speed penalties for ascending or speed gains for descending. The two restrictions on CGF movement make LWACS a simulator with a relatively low fidelity. However, the CGFs in LWACS still represent the basic functions that are required in air combat (e.g., manoeuvring, the use of radar and RWR, firing and evading missiles). Many combinations of these functions are possible in LWACS, making it difficult to design good behaviour models manually. Therefore, we consider LWACS an adequate simulator for our investigation into the use of machine learning for the automatic generation of behaviour models.

A.3 Scripting language

In LWACS, CGF behaviour is defined by scripts. The scripts are written in a custom scripting language. The grammar of this language and the available functions are described in Appendix B.

The scripts are collections of rules. Each CGF is assigned a script. At a rate of 50 Hz, the simulation checks the scripts of all CGFs to determine if any rules should fire. If a rule fires, the actions in the consequence of that rule are executed.

It may occur that multiple rules fire at the same time, e.g., if the rules have overlapping observations in their conditions. In order to provide some control over rule execution in such occurrences, we assign a priority value to each rule. The priority value allows the creators of the rules to specify which rules provide the most urgent or important behaviour. Now, when multiple rules fire, only the rule with the highest priority value is allowed to execute its consequence. In the rare case that multiple rules fire with the same priority value, only the rule that first appears in the script is allowed to execute its consequence.

The CGFs are allowed to maintain a *state* variable. The state is expressed as an alphanumeric identifier. The scripting language can read and write the state variable, so that (1) certain rules can only fire if the CGF is in a particular state, and (2) rules can change the state of the CGF when certain observations are made. The state variable enables sequential scripting, i.e., firing one rule first, and then a second rule, and so on. For instance, we used sequential scripting to create a script that described a patrol route for a CGF. We assigned the value 'patrolling' to the state variable of the CGF. We defined two points in the simulation airspace that the CGF had to patrol, which we call point A and point B. In the script, we included two rules that defined the patrol between points A and B. The first rule was written as "if I am near point A, and my state is 'patrolling', fly towards point B". Vice versa, the second rule was written as "if I am near point B, and my state is 'patrolling', fly towards point A". Additionally, we included a rule that said "if my state is patrolling, and I detect an opponent CGF, change my state to 'engaging'". The result of these rules was that the CGF would fly its patrol between points A and B as long as it did not detect an opponent CGF. Once it detected an opponent CGF, the CGF would no longer consider firing the rules that were written for its patrol, since its state variable did no longer satisfy these rules (i.e., it was not longer set to 'patrolling').

A.4 Scenarios

We developed four two-versus-one scenarios, and four two-versus-two scenarios. In the two-versus-one scenarios, two red CGFs encounter a single blue CGF. In the two-versus-two scenarios, the two reds encounter two blue CGFs. The scripts that governs the behaviour of the blue CGFs in each scenario are presented in Appendix C. Below, we describe the two-versus-one scenarios (Appendix A.4.1) and the two-versus-two scenarios (Appendix A.4.2) in detail.

A.4.1 Two-versus-one scenarios

We developed four two-versus-one scenarios for use in LWACS: (1) the basic scenario, (2) the close range scenario, (3) the evasive scenario, and (4) the mixed scenario. We describe the four scenarios below.

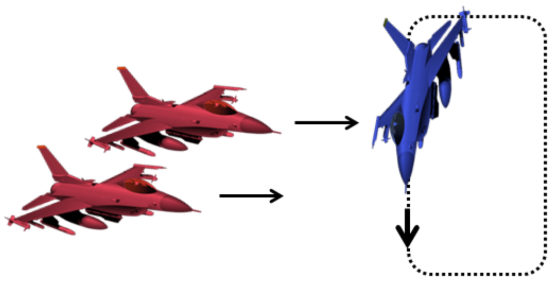


Figure A.2 The initial positions of the CGFs in the four LWACS scenarios. Two red CGFs (left) approach the blue CGF (right), who is flying a CAP.

The basic scenario. One blue CGF flies a combat air patrol (CAP) from north to south (see Figure A.2). When the blue CGF detects an opponent by means of its radar or RWR, it engages that opponent. The blue CGF does not attempt to evade missiles that are fired at it. Two red CGFs fly from east to west and then try to engage the blue CGF.

The close range scenario. The close range scenario is equal to the basic scenario, with one change. The blue CGF only fires missiles from a shorter range. This makes missiles more dangerous, as they need to travel less distance to reach their target, and therefore have a higher P_k on impact.

The evasive scenario. The evasive scenario is equal to the basic scenario, with one change. The blue CGF performs evasive actions when it detects that it is being fired upon.

The mixed scenario. The mixed scenario is a special scenario, because it is a combination of the other three two-versus-one scenarios. Rather than defining new behaviour for the blue, the mixed scenario enables the blue to switch behaviours between encounters. We refer to the mixed scenario as one of the scenarios for convenience. However, in some cases it is helpful to distinguish the mixed scenario from the other three scenarios. In these cases, we refer to the other three two-versus-one scenarios as the *individual* scenarios.

The mixed scenario works as follows. At the start of a run of encounters, one of the three individual scenarios is selected at random, and then used for the first encounter between red and blue. At the end of each encounter, the following individual scenario is selected based on the winner of the encounter. If blue wins the encounter, the same scenario is used in the next encounter. However, if red wins the encounter, the next individual scenario is selected at random. The mixed scenario is inspired by the *consecutive tactic* by Spronck et al. (2006, p. 230).

Compared to only using one of the three individual scenarios, the mixed scenario presents the red team with a *moving target learning problem* (cf. Buşoniu, Babuška and De Schutter, 2010). In other words, red is pressured to come up with behaviour that is effective against an unpredictable opponent. We designed that the mixed scenario to form a more difficult challenge for the red team than only having to find effective behaviour in the three individual scenarios.

The four scenarios have two properties in common. First, the initial positions of the CGFs are the same in all four scenarios. These initial positions are shown in Figure A.2. Second, the four scenarios share their termination criteria. All of the scenarios terminate when either (1) one CGF on either team is hit by a missile, or (2) ten minutes of simulated time has passed.

A.4.2 Two-versus-two scenarios

We developed four two-versus-two scenarios for use in LWACS. The first three of these scenarios are based on the three individual two-versus-one scenarios. In these scenarios, we supplied the single blue CGF with a wingman blue CGF. This made the first blue CGF the lead of the blue two-ship. In all three of the scenarios, the wingman flies the same CAP as the lead, lagging half a pattern behind the lead. The remaining behaviour of the wingman is governed by the same rules as the lead. Thus, during the encounters with the reds, the wingman uses the same tactics as the lead: either (1) attacking the reds without evading (basic scenario), (2) attacking the reds from close range (close range scenario), or (3) attacking the reds while also evading incoming missiles (evasive scenario).

As the fourth scenario, we developed a novel two-versus-two scenario. This scenario is called the lead-trail scenario. We describe this scenario below.

The lead-trail scenario. This scenario is based on the lead-trail tactic that is commonly used by two-ship formations. In this scenario, the blue lead flies head-on towards the red two-ship. The blue wingman flies straight after the blue lead as they approach the reds. When the reds detect the blue lead, the blue lead turns away, with the intention of keeping the attention (viz. a radar lock) of the reds. Then, the blue wingman is able to stay undetected, and then create an opportunity to fire at the reds.

The two-versus-two scenarios have the same two termination criteria as the two-versus-one scenarios.

Appendix B

The LWACS scripting language

In this appendix, we present the LWACS scripting language. The language is used to write the scripts that define the behaviour of the CGFS in LWACS. Below, we first describe the grammar of the scripting language (Appendix B.1). Next, we describe the functions that are available for use in the scripts (Appendix B.2).

B.1 Grammar

Listing B.1 is a formal description of the grammar of the LWACS scripting language in Extended Backus-Naur form. The boolean functions, numerical functions, and action functions are further explained in Section B.2.

Listing B.1 Grammar of the LWACS scripting language.

```
⟨script⟩              ::=  ⟨list-of-rules⟩


⟨list-of-rules⟩       ::=  ⟨rule⟩ end-of-line ⟨list-of-rules⟩
                        |  '#' comment-string end-of-line ⟨list-of-rules⟩
                        |  end-of-line ⟨list-of-rules⟩
                        |  ⟨empty⟩


⟨rule⟩                ::=  ⟨name⟩ [⟨weight⟩] ⟨priority⟩ ⟨condition⟩ '→' ⟨consequence⟩ ';'


⟨name⟩                ::=  '[' identifier ']'


⟨weight⟩                ::  = '[' integer ']'
```

```
⟨priority⟩               :: = '[' integer ']'

⟨condition⟩              :: = ⟨boolean-expression⟩

⟨boolean-expression⟩    ::=  ⟨boolean⟩
                          |  ⟨boolean-function⟩
                          |  'not' ⟨boolean-expression⟩
                          |  '(' ⟨boolean-expression⟩ ')'
                          |  ⟨boolean-expression⟩ ('and' | 'or' | '==') ⟨boolean-expression⟩
                          |  ⟨numerical-expression⟩ ('>' | '<' | '==') ⟨numerical-expression⟩
                          |  'state == ' ⟨state⟩

⟨boolean⟩               ::=  'true' | 'false'

⟨boolean-function⟩      ::=  'isAlive(' ⟨cgf⟩ ')'
                          |  'isRadarMode(' ⟨radar-mode⟩ ')'
                          |  'messageReceived(' ⟨message⟩ ')'
                          |  'missileFlyingAt(' ⟨cgf⟩ ')'
                          |  'missilesLeft'
                          |  'onEvent(' ⟨event⟩ ')'

⟨cgf⟩                   ::=  'ownship'
                          |  'wingman'
                          |  'nearestRadarObservation'
                          |  'nearestRadarWarningReceiverObservation'
                          |  'entity(target)'

⟨event⟩                 ::=  'newRadarObservation'
                          |  'newRadarWarningReceiverObservation'
                          |  'newMissileFlyingAtMe'

⟨team⟩                  ::=  'enemy'

⟨message⟩               ::=  identifier

⟨state⟩                 ::=  identifier
```

⟨*radar-mode*⟩	::=	'searching' | 'track'
⟨*numerical-expression*⟩	::=	⟨*number*⟩
	|	⟨*numerical-function*⟩
	|	⟨*numerical-expression*⟩ ('+' | '–' | '*' | '/') ⟨*numerical-expression*⟩
⟨*number*⟩	::=	integer | float
⟨*numerical-function*⟩	::=	'`countRadarObservations(`' ⟨*team*⟩ '`)`'
	|	'`countRWRObservations(`' ⟨*team*⟩ '`)`'
	|	'`distanceToPoint(`' ⟨*cgf*⟩ '`,`' ⟨*numerical-expression*⟩ '`,`' ⟨*numerical-expression*⟩ '`,`' ⟨*numerical-expression*⟩ '`)`'
	|	'`heading(`' ⟨*cgf*⟩ '`)`'
	|	'`random(`' ⟨*number*⟩ '`,`' ⟨*number*⟩ '`)`'
	|	'`relativeBearing(`' ⟨*cgf*⟩ '`,`' ⟨*cgf*⟩ '`)`'
⟨*consequence*⟩	::=	⟨*action-list*⟩
⟨*action-list*⟩	::=	⟨*action-function*⟩
	|	⟨*action-function*⟩ ⟨*action-list*⟩
⟨*action-function*⟩	::=	'`changeHeading(`' ⟨*numerical-expression*⟩ '`);`'
	|	'`changeState(`' ⟨*state*⟩ '`);`'
	|	'`fireMissile(`' ⟨*cgf*⟩ '`);`'
	|	'`flyTo(`' ⟨*number*⟩ '`,`' ⟨*number*⟩ '`,`' ⟨*number*⟩ '`);`'
	|	'`radarTrackTarget(`' ⟨*cgf*⟩ '`);`'
	|	'`radarSearchTarget;`'
	|	'`sendMessage(`' ⟨*message*⟩ '`,`' ⟨*cgf*⟩ '`);`'
	|	'`skip;`'
	|	'`turn(`' ⟨*numerical-expression*⟩ '`);`'

Three comments regarding the scripting language and its grammar:

- `identifier` represents any alphanumeric word. Dashes are allowed in identifiers, but not as leading or trailing characters.
- `comment-string` represents any comment in natural language.
- `entity(target)` represents the point in space that is the center of the blue CGF's CAP. It can be used in place of a ⟨*cgf*⟩ parameter. In some scripts and rulebases, it is used to explicitly steer the red team towards the blue team at the beginning of a simulation.

B.2 Function descriptions

Below, we describe the boolean functions (Subsection B.2.1), the numerical functions (Subsection B.2.2), and the action functions (Subsection B.2.3).

B.2.1 Boolean functions

isAlive(*target*: ⟨*cgf*⟩)

Returns `true` if *target* is alive, returns `false` otherwise.

isRadarMode(*mode*: ⟨*radar-mode*⟩)

Returns `true` if the CGF's radar is set to *mode*, returns `false` otherwise.

messageReceived(*message*: ⟨*message*⟩)

Returns `true` if the CGF's has received *message*, returns `false` otherwise. The message is consumed.

missileFlyingAt(*target*: ⟨*cgf*⟩)

Returns `true` if a missile is flying at *target*, returns `false` otherwise.

missilesLeft

Returns `true` if there are missiles left in the CGF's inventory, returns `false` otherwise.

onEvent(*event*: ⟨*event*⟩)

Returns `true` if the CGF has been notified of *event*, returns `false` otherwise. The event is consumed.

B.2.2 Numerical functions

countRadarObservations(*team*: ⟨*team*⟩)

Returns the number of CGFs belonging to *team* that are detected by the CGF's radar.

countRadarWarningReceiverObservations(*team*: ⟨*team*⟩)

Returns the number of CGFs belonging to *team* that are detected by the CGF's RWR.

distanceToPoint(*target*: ⟨*cgf*⟩, *x*: ⟨*numerical-expression*⟩, *y*: ⟨*numerical-expression*⟩, *z*: ⟨*numerical-expression*⟩)

Returns the distance (in kilometers) of *target* to the point (x, y, z).

missilesLeft

Returns `true` if there are missiles left in the CGF's inventory, returns `false` otherwise.

B.2.3 Action functions

changeHeading(*heading*: ⟨*numerical-expression*⟩)

Steer the CGF towards heading *heading* (in degrees).

changeState(*state*: ⟨*state*⟩)

Set the CGF's state to *state*.

fireMissile(*target*: ⟨*cgf*⟩)

Fire a missile at *target*.

flyTo(x: ⟨*numerical-expression*⟩, y): ⟨*numerical-expression*⟩, z: ⟨*numerical-expression*⟩)

Steer the CGF towards point (x, y, z).

sendMessage(*message*: ⟨*message*⟩, *target*: ⟨*cgf*⟩)

Send *message* to *target*.

skip(*mode*: ⟨*radar-mode*⟩)

Do nothing (viz. continue flying on the current heading with the current speed).

radarTrackTarget(*target*: ⟨*cgf*⟩)

Set the radar to tracking mode and direct it to track *target*.

radarSearchTarget

Set the radar to search mode.

turn(*offset*: ⟨*numerical-expression*⟩)

Change the CGF's heading by *offset* (in degrees).

Appendix C

Rulebases and scripts

Appendix C contains a listing of the rulebases and scripts that are used in this thesis. The rulebases and scripts are available for download at `http://www.armontoubman.com/phd`. The rules in the rulebases and scripts are formatted as follows:

```
[name] [weight] [priority] condition → consequence
```

Below, we provide an index of the rulebases and scripts that were used in this thesis. Each starred item has been archived as a separate file. For reasons of confidentiality, we are unable to distribute the rulebases and scripts that were used in Chapter 7.

Red team

- Team coordination (Chapter 3)
  - CENT method
    - ∗ Red lead (rulebase)
    - ∗ Red wingman (script)
  - TACIT method
    - ∗ Red lead (rulebase)
    - ∗ Red wingman (rulebase)
  - DECENT method
    - ∗ Red lead (rulebase)
    - ∗ Red wingman (rulebase)
- The AA-REWARD reward function (Chapter 4)
  - Red lead (rulebase)
  - Red wingman (rulebase)
- Transfer of behaviour models between scenarios (Chapter 5)

  - Red lead (rulebase)
  - Red wingman (rulebase)

Blue team

- Two-versus-one scenarios
  - Basic scenario
    - * Blue lead (script)
  - Close range scenario
    - * Blue lead (script)
  - Evasive scenario
    - * Blue lead (script)
  - Mixed scenario
    - * Note: in the mixed scenario, the blue CGF used the scripts from the other three scenarios. Therefore, no specific scripts were made for the blue in the mixed scenario.
- Two-versus-two scenarios
  - Basic scenario
    - * Blue lead (the same script as in the two-versus-one basic scenario)
    - * Blue wingman (script)
  - Close range scenario
    - * Blue lead (the same script as in the two-versus-one close range scenario)
    - * Blue wingman (script)
  - Evasive scenario
    - * Blue lead (the same script as in the two-versus-one evasive scenario)
    - * Blue wingman (script)
  - Lead-trail scenario
    - * Blue lead (script)
    - * Blue wingman (script)

Appendix D

The Fighter 4-Ship simulator

Appendix D describes the Fighter 4-Ship simulator. The Fighter 4-Ship is a research fighter aircraft simulator used by the Netherlands Aerospace Centre NLR (Netherlands Aerospace Centre, 2017b). The simulator allows four human fighter pilots to participate simultaneously in a simulated air combat mission. The purpose of the Fighter 4-Ship is to enable concept development and experimentation in the area of training simulations.

The Fighter 4-Ship consists of (a) four cockpit mock-ups (referred to as the *ships*), and (b) a station for the *instructor*, i.e., the person who controls the operation of the Fighter 4-Ship. This station is accordingly called the instructor operating station (IOS). Below, we describe the hardware of the ships (Appendix D.1), the IOS (Appendix D.2), and the software packages (Appendix D.3) that make up the Fighter 4-Ship.

D.1 The ships

The four ships of the Fighter 4-Ship are modelled after the cockpit of the F-16 fighter jet. Figure D.1 shows a schematic top view of a ship, and Figure D.2 shows a photograph of one of the ships in operation. Each of the four ships is comprised of the following nine items. The items are marked in Figure D.1.

1. A seat.
2. A touchscreen monitor in front of the seat.
3. A physical side stick controller to the right of the seat.
4. A physical throttle to the left of the seat.
5. Physical rudder pedals.

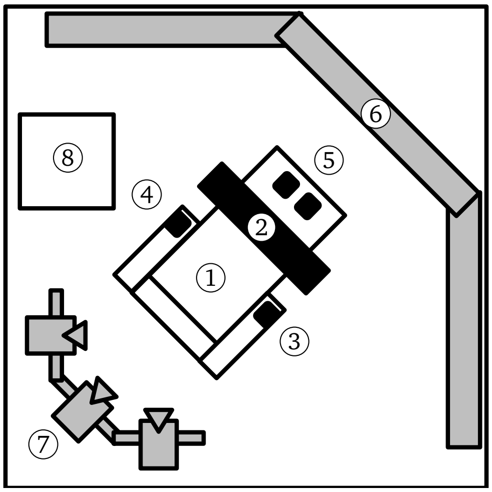


Figure D.1 Schematic top view of a ship.

6. Projector screens that present the participant with (1) the outside view, and (2) an overlayed head-up display (HUD). Two of the ships are equipped with three screens: a left screen, a right screen, and a centre screen. The remaining two ships are equipped with a centre screen only.

7. A projector for each projector screen.

8. Computers that (1) generate the outside visuals which are projected onto the projector screens, (2) control the touchscreen, and (3) handle the voice communications.

9. A headset (not depicted in Figure D.1) for (1) sound effects (e.g., engine noise, warning signals) and (2) voice communication with the instructor and the participants in the other ships over simulated radio channels.

The seat, stick, throttle, and pedals are replicas of the equipment in a real-world F-16 cockpit. The touchscreen monitor shows representations of the F-16 consoles and controls. The representations include two MFDs, viz. square screen which provide overviews of the aircraft's situation, and allow the pilot quick access to various functions by means of hierarchical menus.

The ships are *fixed-base*, viz. they do not provide any motion effects. Each of the four ships is located in a separate but adjacent room. The aircraft noise that is to be heard over the headsets prevent any verbal communication between the pilots, other than using the simulated radio channels. Furthermore, because of the physical separation, a participant in one of the ships is unable to see any of the other ships.

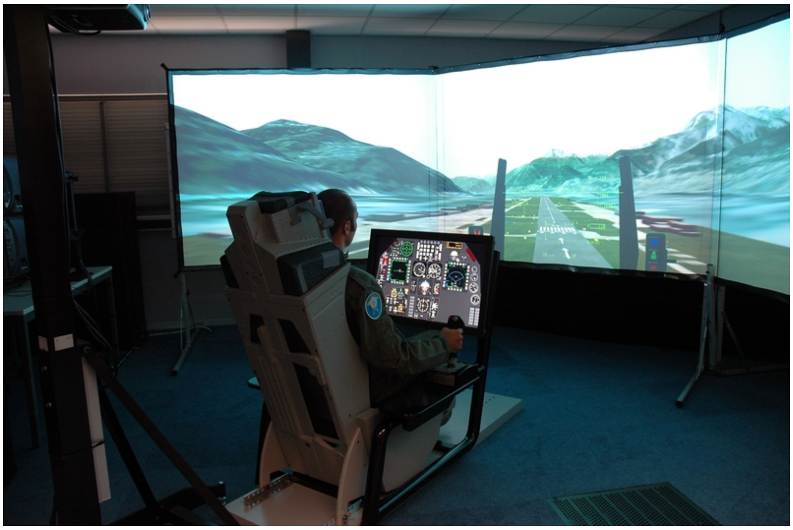


Figure D.2 Photograph of a ship being operated by a participant.

D.2 The instructor operating station

The simulation is controlled from the instructor operating station (IOS). This station provides the instructor with the capability to (1) control the operation of the individual ships, (2) start and stop scenarios, (3) add and remove CGFS, (4) assign behaviour models to CGFS, (5) communicate with the participants in the ships, (6) view and record the MFDS of the four ships in order to monitor the actions of the participants, and (7) view and record the simulated environment and all entities within it. The IOS is comprised of the following two items.

1. A desktop computer with five monitors.
   1. A monitor that displays EUROSIM, the software that starts/stops/pauses the operation of each of the four ships. The functions of EUROSIM and the other software that is displayed on the monitors are explained in Appendix D.3.
   2. A monitor that displays STAGE, the software that creates the CGFS.
   3. A monitor that displays SMART BANDITS, the software that executes the behaviour models of the CGFS.
   4. A monitor that displays a video stream of the MFDS of the four cockpits.
   5. A monitor that displays the debrief software that records the simulated environment.
2. A headset for voice communication with the participants in the ships. Additionally, this headset monitors all simulated radio channels. So, the instructor at the IOS remains informed of all communication that happens among participants.

The use of five monitors allows the instructor to easily view and access all functions that are relevant to controlling and monitoring the simulations.

D.3 Software packages

The Fighter 4-Ship runs on four software packages: (1) EUROSIM, (2) STAGE, (3) SMART BANDITS, and (4) PCDS. We describe the four software packages below.

EUROSIM. EuroSim (2017) is a simulation framework that provides interfaces for air and space simulations. In the Fighter 4-Ship, EUROSIM executes the models of the flight dynamics the aircraft that the four ships represent. Additionally, EUROSIM simulates the avionics (e.g., the radar) of the ships. EUROSIM provides a GUI by which the operation of individual ships can be controlled and inspected by the instructor.

STAGE. STAGE (Presagis, 2012) is a “simulation development framework” that provides an environment for building and executing scenarios. The instructor can select CGFS from a database, and place them into the simulated world. STAGE includes a basic behaviour editor, which is currently not used in the Fighter 4-Ship. Finally, STAGE provides a GUI by which the state of CGFS can be inspected and manipulated during simulations.

SMART BANDITS. SMART BANDITS (Netherlands Aerospace Centre, 2017a) is software by which CGF behaviour can be modelled as a FSM. In SMART BANDITS, the states and transitions that make up FSMS can be combined by means of drag-and-drop functionality. This way, it allows professionals such as training specialists to create behaviour models without any explicit programming knowledge. During simulations, the behaviour of CGFS can be inspected and manipulated.

PCDS. PCDS (Naval Air Systems Command, 2010) provides record and playback facilities for simulation environments. It is intended to aid debriefing, i.e., the meeting after a simulation session when the instructor reviews the simulation with the participants (e.g., to identify learning opportunities). Video files (e.g., recordings of the MFD streams) can be connected to the playback of simulations, so that the internal situation in each ship can be seen when a simulation is reviewed.

The four software packages communicate with each other and with the four ships by means of the distributed interactive simulation (DIS) standard (IEEE, 2012). This standard defines a common format for the exchange of information regarding, e.g., the location and status of CGFS. This information is used by the four software packages to determine, e.g., how a CGF should be visualised.

D.4 Dynamic scripting in the Fighter 4-Ship

At the beginning of our research, the Fighter 4-Ship had no machine learning capabilities. Therefore, the integration of a machine learning technique such as dynamic scripting required us to determine the placement of this technique between the Fighter 4-Ship's software packages. In other words, we needed to consider the architecture of the Fighter 4-Ship's software.

In the Fighter 4-Ship, STAGE is the software package that generates the CGFS. STAGE exposes an API by which other software packages can (1) read the state and observations of its CGFS, (2) direct the actions of the CGFS, and (3) start and stop predefined scenarios. SMART BANDITS controls the CGFS in STAGE by means of this API. In SMART BANDITS, the state and observations of a CGF are read from the API, and then serve as input for a behaviour model. The behaviour model then outputs the actions that the CGF should take. The actions are sent back to the CGF in STAGE via the API.

To implement dynamic scripting, we require software with a function similar to how SMART BANDITS executes behaviour models for the CGFS in STAGE. However, compared to SMART BANDITS, we require two additional functions: (1) running the dynamic scripting algorithm to generate new behaviour models (particularly, the rule-based FSM models that are described in Appendix E), and (2) control over the scenarios that are run in the simulator.

For our research, we developed a new program which combines the three functions. We call this program STAGEDS. STAGEDS provides us with two capabilities: (1) to let CGFS learn by means of the dynamic scripting algorithm, and (2) to halt the learning process and only execute the learned behaviour models. We make extensive use of these two capabilities in Chapter 7. Ideally, in the future, these capabilities will be built into the software that is currently used for the design and execution of CGF behaviour models (i.e., SMART BANDITS).

Appendix E

Generating finite-state machines

Appendix E discusses the generation of finite-state machines by means of dynamic scripting.

In the automated simulations that were performed in LWACS, the rules in the scripts of CGFS produced behaviour in an *ad-hoc* manner: whenever the condition of a rule was met, the actions in that rule's consequence would immediately be executed. As we have shown in the Chapters 3 to 5, the behaviour which is produced in this manner is effective for CGFS in automated simulations. However, since then, we have been informed that this way of producing behaviour is not completely suitable for human-in-the-loop simulations.

During simulations, scripts are sensitive to small changes in the environment. Some of these small changes should have no effect on the behaviour of a CGF, but may still cause different rules to fire. As an example, assume that a CGF is under attack, and a defensive rule fires which causes the CGF to make a defensive manoeuvre. However, during this manoeuvre, the CGF simultaneously detects another opponent by means of its radar, and an offensive rule is prompted to fire. At this point, the CGF is still in danger from the first opponent, and should continue with its defensive manoeuvre instead of preparing an attack on the second opponent. Moreover, from an external point of view, such sudden jumps between rules make the CGF appear erratic and indecisive, i.e., not precisely the way how a formidable adversary should behave. To remedy such a jump in behaviour, the CGF needs to be able to remember what it is doing and why. In other words, the CGF needs a concept such as a state in which it feels situated, so that the CGF can select and display behaviour that is relevant for that state.

Returning to the topic of representation, in Chapter 2, we mentioned FSMS as one of the executable forms that a behaviour model can take. The management of states and the behaviours therein, e.g., the states and behaviours of a CGF, is the prime function of FSMS. Furthermore, since FSMS are comprised of graph-like structures, they are highly suitable for graphical representation as demonstrated in, e.g., the SMART BANDITS program. Consequently, we have a preference to generate CGFS with dynamic scripting.

Below, we show how the states and transitions that make up an FSM can be readily expressed in the form of rules (Appendix E.1). By expressing states and transitions in the form of rules, dynamic scripting is able to combine the states and transitions into new FSMS, in the same way that dynamic scripting combines "regular" if-then rules into scripts. However, at this point we foresee that the dynamic scripting algorithm needs to be modified so that it can combine these rules into functioning FSMS. We present the exact modifications that we make to the original dynamic scripting algorithm (Appendix E.2). Thereafter, we conclude the appendix by a summary (Appendix E.3).

E.1 Expressing finite-state machines as rules

In this section, we explain how FSMS can be expressed as rules. We do so by means of the following example. Consider a CGF that performs a patrol between two points, namely point A and point B. Patrolling between point A and point B should take place until the CGF detects some hostile CGF. At that point, it should engage the hostile CGF. Then, when the hostile CGF is defeated, it should return to its patrol.

The FSM for the example above is shown in Figure E.1. This FSM contains two states: (1) the *Patrol* state, shown at the top, and (2) the *Engage* state, shown at the bottom. The FSM has two transitions between the states. The CGF starts in the *Patrol* state, and flies between point A and point B. When the CGF is in the *Patrol* state and detects a hostile CGF, a transition from the *Patrol* state to the *Engage* state takes place. While in the *Engage* state, the CGF performs some actions with the goal of defeating the hostile CGF. Once the hostile CGF is defeated in some way, our CGF transitions back from the *Engage* state to the *Patrol* state, and then continues its patrol.

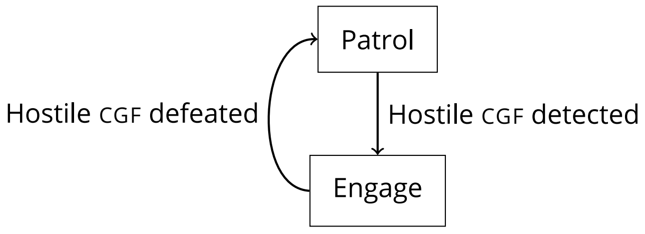


Figure E.1 A basic example of an FSM as a behaviour model.

We translate the FSM from Figure E.1 into rules as follows. The resulting rules are shown in Listing E.1. In the first rule (shown on line 1–3), we define the behaviour that the CGF should perform when it is in the *Patrol* state. We call this type of rule a *state rule*. When the CGF is in the *Patrol* state, it should fly between point A and point B. We capture this behaviour in the rule by introducing control statements (e.g., *if/then/else* or *while*) into the consequence of the rule.

Listing E.1 The FSM from Figure E.1 expressed in the form of rules.

```
1 in_state(Patrol) →
2     if near(point_A) then fly_towards(point_B)
3     else if near(point_B) then fly_towards(point_A);
4 in_state(Patrol) and hostile_CGF_detected() →
     change_state_to(Engage);
5 in_state(Engage) → attack(detected_hostile_CGF);
6 in_state(Engage) and is_defeated(detected_hostile_CGF) →
     change_state_to(Patrol);
```

In the second rule (line 4) we define the transition from the *Patrol* state to the *Engage* state. We call this rule a *transition rule*. When the CGF is in the *Patrol* state and it detects a hostile CGF, it transitions to the *Engage* state.

The third rule (line 5) defines the behaviour for the *Engage* state. Here, we tell the CGF to attack the detected hostile CGF. Afterwards, when the hostile CGF is defeated, the fourth rule fires (line 6) and the CGF returns to its patrol between point A and point B.

The resulting rules can now be stored in a rulebase, which serves as the input for dynamic scripting. A script with state rules and transition rules then becomes the implementation of a FSM. By creating variations of the state rules and the transition rules, and storing these variations in the rulebase as well, dynamic scripting is able to explore the space of possible FSMs.

E.2 The modified dynamic scripting algorithm

In the original description of dynamic scripting (Spronck et al., 2006), rules are selected in a probabilistic manner, under the assumption that all rules are valid choices for inclusion in a script. However, when the rulebase is filled with state rules and transition rules, a problem arises: namely, the rules that are selected for inclusion in a generated script must together form a valid FSM. We define an invalid FSM as one that contains unreachable states.

To help dynamic scripting create valid FSMs, we restrict the generation of FSMs to those that follow a specific template. We define a template to be a collection of (1) states and (2) transitions between two states, without any specification of the behaviour that these states and transitions represent. Henceforth, we use the term *element* to refer to either a state or a transition. For example, the FSM shown in figure 7.2 has four elements: (1) the Patrol state, (2) the Engage state, (3) the transition from Patrol to Engage, and (4) the transition from Engage to Patrol. The use of a template allows us to define the structure that a FSM should follow, thereby providing a guarantee that the FSM is valid. However, templates make it possible to leave the choice of *implementation*

Listing E.2 Modified script generation algorithm.

```
1  # input:    rulebase, fsm_template
2  # output:   script
3  script = []
4  for element in fsm_template:
5      sum_of_weights = 0
6      candidate_rules = []
7      for rule in rulebase:
8          if rule.is_implementation_of(element):
9              candidate_rules.append(rule)
10             sum_of_weights += rule.weight
11     if sum_of_weights == 0:
12         selected_rule = random.choice(candidate_rules)
13         script.append(selected_rule)
14     else:
15         selected_rule = roulette_wheel(candidate_rules)
16         script.append(selected_rule)
17 return script
```

(i.e., the actual behaviour for states, and the specific conditions on which transitions are made) to dynamic scripting.

We replace the original script generation algorithm (Spronck et al., 2006, p. 224, Algorithm 1) by the new algorithm shown in Listing E.2. The new algorithm takes as input a rulebase and an FSM template. First, an empty script is created (line 3). Next, for each element in the FSM template, an implementation is selected for inclusion in the script (line 4–16). This is done per element by first filtering out the candidate rules in the rulebase that are an implementation of that element (line 8–10). From these candidate rules, an implementation is selected by means of the original weight-proportionate roulette wheel selection (as explained in Subsection 2.3.3).

E.3 Summary

In this appendix, we enabled the dynamic scripting algorithm to generate FSMs. In some cases, the use of scripts as a behaviour model allows CGFs to jump erratically between behaviours. The use of FSMs provides the CGFs with a sense of state, so that the CGFs will only display behaviour that is relevant to the state they are in.

Two steps were required for generating FSMs by means of dynamic scripting. First, we translated the states and the transitions between the states to the form of rules. This way, the states and transitions can be treated as rules in the rulebase of dynamic scripting. Second, we altered the mechanism by which the dynamic scripting algorithm selects rules from its rulebase, so that the generated FSMS follow a pre-specified template. The use of this templates ensures that all generated FSMS are valid, i.e., they contain states and the appropriate transitions between the states. By enabling the dynamic scripting algorithm to generate FSMS, we gain the benefits of FSMS (i.e., the concept of state and the possibility of graphical representation) while maintaining the benefits of dynamic scripting (i.e., easily inspectable rules and rulebases, and diverse combinations of behaviour from the rulebase).

Appendix F

The Assessment Tool for Air Combat CGFs

In this appendix, we present our implementation of the ATACC questionnaire. We implemented the ATACC as a single-page form. This form was used by expert assessors to assess the behaviour of CGFS in human-in-the-loop simulations (see Chapter 7).

On the form, we included (a) the nine rating items of the ATACC, and (b) four fields for additional information. The nine rating items of the ATACC are discussed in Section 6.4. The four fields are labelled as follows: (1) *Tactical*, (2) *Set code*, (3) *Start time*, and (4) *Operational status*. Below, we explain these four fields.

Tactical The tactical is a personal code name that is used for both operational security and convenience. We included the tactical of the assessors to be able to quickly refer to specific forms that were filled in.

Set code In the preparation of the recorded human-in-the-loop simulations, we assigned a code consisting of a letter and a number to each specific encounter (a.k.a. a *setup* or *set* by the assessors) that was recorded. Each assessor was provided with a sheet that showed all the set codes in an individually randomised order. The assessors were instructed to (1) take the codes in the order that they were listed on the sheet, then (2) use each code to look up the encounter belonging to that set in the PCDS program that was running on their laptop, after which they were to (3) take an empty form, note down the code, and assess the behaviour of the CGFS in the recording. The set code field on the form acted as a control to ensure that the assessors viewed the recorded encounters in the order that was assigned to them.

Start time When an assessor used a set code to look up a recorded encounter in the PCDS program, PCDS displayed a time index for the start of that encounter. The assessors were instructed to write down that time index in the start time field. The start time field acted as a control that allowed us to determine whether the assessors had correctly selected and viewed the encounter which was indicated by the set code.

Operational status The operational status of the assessors indicates their level of experience. The seven options that are provided (i.e., wingman, 2FL, 4FL, MC, IP, WIP, and TIP) refer to the qualifications that can be obtained by the assessors.

The following page shows the exact form that was used in Chapter 7.

Assessment Tool for Air Combat CGFs	**Tactical:**	**Set code:**
		Start time:
	Operational status:	Wingman / 2FL / MC / WIP / 4FL / IP / TIP

	Never	Rarely	Sometimes	Often	Always
Red air forced blue air to change their tactical plan.	O	O	O	O	O
Red air forced blue air to change their shot doctrine.	O	O	O	O	O
Red air was within factor range.	O	O	O	O	O
Blue air was able to fire without threat from red air.	O	O	O	O	O
Red air acted on blue air's geometry.	O	O	O	O	O
Red air acted on blue air's weapons engagement zone.	O	O	O	O	O
Red air flew with kinematic realism.	O	O	O	O	O
Red air's behaviour was intelligent.	O	O	O	O	O

	Strongly disagree	Disagree	Undecided	Agree	Strongly agree
Red air's behaviour tested blue air's tactical air combat skills.	O	O	O	O	O

Summary

By training with virtual opponents known as computer generated forces (CGFS), trainee fighter pilots can build the experience necessary for air combat operations, at a fraction of the cost of training with real aircraft. In practice however, the variety of CGFS is not as wide as it can be. This is largely due to a lack of behaviour models for the CGFS. The lack motivated me to design and improve air combat simulations. In this thesis we investigate to what extent behaviour models for the CGFS in air combat training simulations can be *automatically* generated, by the use of machine learning.

The domain of air combat is complex, and machine learning methods that operate within this domain must be suited to the challenges posed by the domain. In Chapter 1, we identify five challenges that must be met before newly generated behaviour models can effectively be applied in training simulations. These are: (A) producing team coordination, (B) computationally evaluating CGF behaviour, (C) efficient reuse of acquired knowledge, (D) validating generated behaviour models, and (E) generating accessible behaviour models.

From the above motivation for the research, together with the five challenges, we derive the following problem statement: *To what extent can we use dynamic scripting to generate air combat behaviour models for use in training simulations, in such a way that the five challenges of generating air combat behaviour models are met?* The problem statement mentions the use of the dynamic scripting algorithm. This algorithm produces human-readable behaviour models, and thus enables us to meet challenge E. Based on the remaining four challenges, we formulate five research questions that we investigate in the remainder of the thesis.

In Chapter 2, we present background information on the process by which behaviour models are created today. Furthermore, we introduce (a) machine learning, and (b) the dynamic scripting algorithm in particular. Additionally, we review earlier work on the subject of generating air combat behaviour models by means of machine learning.

In Chapter 3, we investigate research question 1: *To what extent can we generate air combat behaviour models that produce team coordination?* Today, the smallest unit that performs air combat missions is the *two-ship*, consisting of a *lead* and a *wingman* aircraft. To succeed in their missions, the lead and the wingman in a two-ship need to carefully coordinate their actions. Therefore, such coordination should be reflected in the behaviour models of a two-ship

of CGFs. We define three coordination methods within the rule-based framework of dynamic scripting: (1) a decentralised coordination method without communication called TACIT, (2) a centralised coordination method with communication called CENT, and (3) a decentralised coordination method with communication called DECENT. Next, we perform three series of automated simulations. In each series, we use dynamic scripting to generate behaviour models for a two-ship that engages a pre-programmed opponent while coordinating by one of the coordination methods. We find that each of the three methods leads to a flexible division of roles between the CGFs. Out of the three methods, the coordination produced by the CENT method resulted in the most effective behaviour that reached the highest win rates. Based on our research, we may conclude that by means of dynamic scripting, we are able to (a) generate multiple forms of team behaviour, and (b) easily inspect the roles assumed by the team members.

In Chapter 4, we investigate research question 2: *To what extent can we improve the reward function for air combat CGFs?* The reward function is an essential part of dynamic scripting. It evaluates the desirability of the behaviour produced by the behaviour models that are generated, and then produces a reward signal that stimulates the dynamic scripting algorithm to improve the models in a next iteration. A commonly used reward function is the *binary* reward function: a reward signal of 1 is provided if the CGFs win a simulated encounter (i.e., show desirable behaviour) using the generated behaviour model, otherwise a reward signal of 0 is provided. However, because this reward signal is both *sparse* (i.e., a CGF has to display exactly the right behaviour before a reward is obtained) and *unstable* (i.e., non-determinism in the CGF's environment may cause the same behaviour to lead to different results), it is possible that more desirable behaviour can be achieved by using a more suitable reward function. We develop two new reward functions for use in the air combat domain: DOMAIN-REWARD which is aimed at making the rewards less sparse, and AA-REWARD which is aimed at making the rewards stable. Both are tested in automated simulations. From the results we may conclude that while DOMAIN-REWARD fails to improve the behaviour of the CGFs over the use of a binary reward function, the use of AA-REWARD leads to a 12.6% increase in win rates.

In Chapter 5, we investigate research question 3: *To what extent can knowledge built with dynamic scripting be transferred successfully between CGFs in different scenarios?* The behaviour models generated by the dynamic scripting algorithm contain knowledge about air combat situations. For instance, in Chapter 3 and Chapter 4, we used dynamic scripting to generate behaviour models for use by a two-ship in a two-versus-one scenario. We hypothesise that the knowledge contained in these models is to some extent reusable between different scenarios. We place a two-ship of CGFs in scenarios in which they have to learn to defeat two opponents, and then generate behaviour models for the two-ship. We do so *twice*: once, the two-ship has to learn to defeat the two opponents with a *tabula rasa*; the next time, the algorithm that generates the behaviour models for the two-ship is initialised with the behaviour models (in the form of weighted rules) that were generated in earlier two-versus-*one* scenarios. In each of the two-versus-two scenarios, we find that the two-ship using the transferred knowledge learns more effective behaviour than the other two-ship. Furthermore, they take less time to reach their highest level of performance.

In Chapter 6, we investigate research question 4: *How should we validate machine-generated air combat behaviour models for use in training simulations?* Validation is an important step in the development of behaviour models, since it provides a structured way to determine whether the models are useful with regards to their intended purpose (in our case, training simulations). However, there is no *one-size-fits-all* to the validation of behaviour models. Therefore, in Chapter 6, we develop a new validation procedure specifically for machine-generated air combat behaviour models. In brief, our procedure consists of three steps. The first step is recording human-in-the-loop simulations, in which human participants engage CGFS that are controlled by a sample of either (a) behaviour models that have been manually designed by human professionals, or (b) newly generated behaviour models. The second step is a structured assessment of the behaviour displayed by the CGFS in the recordings. The assessment is performed by expert assessors, by means of the newly developed Assessment Tool for Air Combat CGFS (ATACC). The third step is the use of the TOST method (two one-sided t-tests) to determine whether the assessments of the behaviour produced by the manually designed behaviour models are statistically equivalent to the assessments of the behaviour produced by the machine-generated behaviour models. If so, we consider the generated behaviour models to be valid for application within training simulations.

In Chapter 7, we investigate research question 5: *To what extent are air combat behaviour models generated by means of dynamic scripting valid for use in training simulations?* We apply the validation procedure that is developed in Chapter 6 to a set of newly generated behaviour models. As a baseline, we use a set of manually designed behaviour models that has been used in real-world training simulations. We perform human-in-the-loop simulations in which Royal Netherlands Air Force (RNLAF) fighter pilots engage CGFS controlled by the behaviour models. The behaviour displayed by the CGFS in the simulations is assessed by instructor pilots, by means of the ATACC. On the ATACC, the assessors rate the occurrence of nine examples of behaviour. Between the CGFS using the manually designed behaviour models and the generated behaviour models, six out of the nine examples of behaviour are rated as occurring in an equivalent manner. Based on this result, we can neither conclude to a complete validity of the generated behaviour models, nor to a non-validity. However, since the literature advises us to recognise degrees of success, we may conclude that our behaviour models are valid to a moderate extent.

In Chapter 8, we conclude the thesis by summarising the answers given earlier to the five research questions and the problem statement. Our research shows that dynamic scripting greatly facilitates the automatic generation of air combat behaviour models, while being sufficiently flexible to be moulded into answers to the challenges. However, ensuring the validity of the newly generated behaviour models remains to be a point of attention for future research.

Samenvatting

Het trainen met virtuele tegenstanders (ook wel *computer generated forces* of CGFS genoemd) geeft gevechtspiloten de gewenste ervaring voor vliegoperaties, zonder de hoge kosten die het trainen met echte vliegtuigen met zich mee brengt. In de praktijk zijn deze CGFS echter niet zo veelzijdig als ze zouden kunnen zijn. Dit komt vooral door een gebrek aan realistische gedragsmodellen. Het ontbreken van gedragsmodellen motiveerde mij om *air combat training* simulaties te ontwerpen en te verbeteren. In het proefschrift onderzoeken we in hoeverre het mogelijk is om de gedragsmodellen voor CGFS in een luchtgevecht automatisch te genereren, met behulp van *machine learning* (machinaal leren).

Het domein van het luchtgevecht is complex. Daarom is het belangrijk dat de *machine learning* methodes die binnen dit veld ingezet worden, opgewassen zijn tegen de uitdagingen van dit domein. In hoofdstuk 1 identificeren we vijf uitdagingen die nieuw gegenereerde gedragsmodellen het hoofd moeten bieden, om bruikbaar te zijn binnen trainingssimulaties. Dit zijn: (A) het produceren van teamcoördinatie, (B) de computationele evaluatie van CGF-gedrag, (C) het efficiënt hergebruiken van opgebouwde kennis, (D) het valideren van gegenereerde gedragsmodellen en (E) het genereren van gedragsmodellen die toegankelijk en leesbaar zijn voor mensen.

Aan de hand van de bovenstaande motivatie en de vijf uitdagingen, komen wij tot de volgende probleemstelling: *In hoeverre is het mogelijk om* dynamic scripting *te gebruiken om gedragsmodellen te genereren voor CGFS in een luchtgevecht, waarbij ingespeeld wordt op de vijf uitdagingen van gegenereerde gedragsmodellen?* In de probleemstelling wordt gesproken over het *dynamic scripting* (dynamisch schrijf-) algoritme. Dit algoritme produceert leesbare gedragsmodellen en komt zo tegemoet aan uitdaging E. Op basis van de overige vier uitdagingen formuleren we vijf onderzoeksvragen die we in het proefschrift verder gaan onderzoeken.

In hoofdstuk 2 presenteren we achtergrondinformatie over het proces waarmee gedragsmodellen tegenwoordig geproduceerd worden. Bovendien introduceren we (a) *machine learning* en (b) het *dynamic scripting* algoritme in het bijzonder. Daarbij kijken we terug op eerder werk over het genereren van gedragsmodellen voor luchtgevechten met behulp van *machine learning*.

In hoofdstuk 3 onderzoeken we onderzoeksvraag 1: *In hoeverre kunnen we gedragsmodellen voor CGFS in een luchtgevecht genereren waarbij sprake is van teamcoördinatie?* Vandaag de dag is het *two-ship* (een combinatie van twee vliegtuigen, ook bekend als het *tweetje*) de kleinste eenheid die gevechtsmissies in de lucht onderneemt, bestaande uit een *lead* (leider) en een *wingman* (volgvlieger). Om in hun missies te slagen, moeten de *lead* en *wingman* in een *two-ship* nauwgezet samenwerken. Daarom is het belangrijk dit soort samenwerking terug te zien in de gedragsmodellen die worden gegenereerd voor een *two-ship* bestaande uit CGFS. Binnen het raamwerk van het *dynamic scripting* algoritme, definiëren we drie methodes voor dergelijke coördinatie: (1) een gedecentraliseerde methode zonder communicatie tussen de CGFS genaamd TACIT, (2) een gecentraliseerde methode met communicatie genaamd CENT en (3) een gedecentraliseerde methode met communicatie genaamd DECENT. Hierna voeren we drie series van geautomatiseerde simulaties uit. In elke serie gebruiken we *dynamic scripting* om gedrag voor een *two-ship* te genereren dat tegen een voorgeprogrammeerde tegenstander moet vechten, met gebruik van één van de genoemde coördinatiemethodes. Het blijkt dat elk van de drie methodes tot een flexibele rolverdeling tussen de CGFS leidt. Van de drie methodes levert de coördinatie die CENT produceert het meest effectieve gedrag met het hoogste aantal gewonnen ontmoetingen. Gebaseerd op ons onderzoek mogen we concluderen dat we met behulp van *dynamic scripting* in staat zijn om (a) verschillende vormen van team gedrag te genereren en (b) de rolverdeling die binnen een team is aangenomen goed te herkennen.

In hoofdstuk 4 bekijken we onderzoeksvraag 2: *In hoeverre kunnen we het beloningsmechanisme voor CGFS in een luchtgevecht verbeteren?* Het beloningsmechanisme (*reward function* in het Engels) is een essentieel deel van *dynamic scripting*. Het evalueert de wenselijkheid van het gedrag dat voortvloeit uit de gegenereerde gedragsmodellen en produceert vervolgens een belonend signaal dat het *dynamic scripting* algoritme stimuleert om de gedragsmodellen bij de volgende poging te verbeteren. Een veel gebruikt beloningsmechanisme is het binaire beloningsmechanisme (in het proefschrift BIN-REWARD genoemd): daarbij krijgt het algoritme een beloning van 1 als de CGFS een gesimuleerd gevecht hebben gewonnen (gewenst gedrag) aan de hand van de gegenereerde gedragsmodellen en 0 als er is verloren. In het gebruik blijkt echter dat een beloning bij dit mechanisme zowel schaars (een CGFS moet precies het juiste gedrag vertonen om een beloning te kunnen verkrijgen) als onvoorspelbaar (onverwachte gebeurtenissen in de omgeving van de CGFS kunnen ervoor zorgen dat hetzelfde gedrag een andere uitkomst heeft) is. Daarom is het goed mogelijk dat een ander beloningsmechanisme meer gewenst gedrag oplevert. Wij hebben daarop twee nieuwe beloningsmechanismen ontwikkeld voor het gebruik in het luchtgevechtsdomein: DOMAIN-REWARD, dat zich richt op het minder schaars maken van beloningen en AA-REWARD, dat gericht is op het voorspelbaarder maken van de beloningen. Beide mechanismen hebben wij getest in geautomatiseerde simulaties. Uit de resultaten maken wij op dat het de DOMAIN-REWARD niet gelukt om de resultaten van het binaire beloningsmechanisme te verbeteren. Het gebruik van AA-REWARD leidt er echter toe dat er 12,6% vaker wordt gewonnen door de CGFS.

In hoofdstuk 5 bespreken we onderzoeksvraag 3: *In hoeverre kan kennis die verkregen is door* dynamic scripting *succesvol uitgewisseld worden tussen CGF*s *in verschillende situaties?* De gedragsmodellen die gegenereerd worden door het *dynamic scripting* algoritme bevatten kennis over verschillende situaties in luchtgevechten. In hoofdstuk 3 en 4 gebruikten we *dynamic scripting* bijvoorbeeld om gedragsmodellen te genereren voor *two-ships* in twee-tegen-één-scenario's. Wij verwachten dat de kennis in deze gedragsmodellen tot op zekere hoogte in andere scenario's hergebruikt kan worden. We plaatsen een *two-ship* van CGFS in scenario's waarin ze moeten leren om twee vijanden te verslaan. Daarbij gebruiken we *dynamic scripting* om gedragsmodellen voor het *two-ship* te genereren. Dit doen we twee keer. De eerste keer moet het *two-ship* zonder voorkennis leren om de vijanden te verslaan. De tweede keer krijgt het algoritme voorkennis mee, in de vorm van de gedragsmodellen die gegenereerd zijn in een eerder twee-tegen-één-scenario. In alle twee-tegen-twee-scenario's blijkt het *two-ship* met voorkennis effectiever gedrag te vertonen dan het *two-ship* zonder voorkennis. Bovendien wordt dit effectievere gedrag sneller bereikt.

In hoofdstuk 6 bekijken we onderzoeksvraag 4: *Hoe moeten we machine-gegenereerde gedragsmodellen voor luchtgevechtstrainingen valideren?* Validatie is een belangrijke stap in de ontwikkeling van gedragsmodellen, aangezien het een gestructureerde manier biedt om de bruikbaarheid van de modellen te beoordelen (in ons geval voor trainingsdoeleinden). Er is echter geen standaardmanier om alle verschillende soorten gedragsmodellen te valideren. Daarom ontwikkelen we in hoofdstuk 6 een nieuwe validatie methode, die specifiek bedoeld is voor computer-gegenereerde gedragsmodellen voor luchtgevechten. De procedure bestaat uit drie stappen. De eerste stap is het opnemen van *human-in-the-loop* simulaties (vrij vertaald: simulaties met mensen in het terugkoppelingsproces), waarbij menselijke deelnemers het opnemen tegen CGFS die aangestuurd worden op basis van (a) gedragmodellen die gemaakt zijn door menselijke professionals of (b) nieuw gegenereerde gedragsmodellen. De tweede stap bestaat uit een gestructureerde beoordeling van het gedrag van de CGFS. Deze beoordeling wordt uitgevoerd door experts in het veld, met behulp van een nieuw ontwikkeld scoreformulier voor het gedrag van CGFS in een luchtgevecht. Dit scoreformulier noemen wij de ATACC. De derde stap is het gebruik van de TOST (*two one-sided t-tests*, oftewel twee eenzijdige t-tests) methode om te bepalen of de beoordelingen van het gedrag dat voortkomt uit de eerder gemaakte gedragsmodellen statistisch gelijkwaardig zijn aan de beoordelingen van het nieuwe gedrag. Als dit zo is, beschouwen we de computer-gegenereerde gedragsmodellen als valide voor het gebruik in trainingssimulaties.

In hoofdstuk 7 bespreken we onderzoeksvraag 5: *In hoeverre zijn gedragsmodellen die gegenereerd zijn met* dynamic scripting *valide voor het gebruik in trainingssimulaties?* We passen de valideringsmethode uit hoofdstuk 6 toe op een groep nieuw gegenereerde gedragsmodellen. Als ijkpunt gebruiken we een selectie van mens-gegenereerde gedragsmodellen die gebruikt zijn voor daadwerkelijke trainingssimulaties. We draaien *human-in-the-loop* simulaties waarin vliegers van de Koninklijke Luchtmacht het opnemen tegen CGFS die aangestuurd worden door gegenereerde gedragsmodellen. Het vertoonde gedrag wordt beoordeeld door vlieginstructeurs, met behulp van de ATACC. Tijdens de beoordeling wordt er gelet op het voorkomen van negen soorten gedragingen. Vervolgens worden de beoordelingen van de mens-gegenereerde gedragingen en de computer-gegenereerde gedragingen vergeleken. Het blijkt dat zes van de negen soorten gedragingen gelijkwaardig scoren. Op basis van dit resultaat mogen we de computergegenereerde modellen niet als volledig valide beschouwen. De literatuur adviseert ons echter om verschillende gradaties van validiteit te erkennen. We beschouwen de nieuw gegenereerde gedragsmodellen daarom als eniger mate valide.

In hoofdstuk 8 ronden we dit proefschrift af met het geven van een samenvatting van de antwoorden op de vijf onderzoeksvragen en het antwoord op de probleemstelling. Ons onderzoek toont aan dat *dynamic scripting* het geautomatiseerd genereren van gedragsmodellen voor luchtgevechten faciliteert, terwijl het flexibel genoeg blijkt om de uitdagingen van het luchtgevechtsdomein het hoofd te bieden. De validiteit van de nieuw gegenereerde gedragsmodellen blijft een aandachtspunt in komende onderzoeken.

List of publications

The work that is presented in this thesis is based on the following publications.

1. A. Toubman, J. J. Roessingh, P. Spronck, A. Plaat and H. J. Van den Herik (2014a). Dynamic Scripting with Team Coordination in Air Combat Simulation. In: *Modern Advances in Applied Intelligence: 27th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2014, Kaohsiung, Taiwan, June 3-6, 2014, Proceedings, Part I*. Ed. by M. Ali, J.-S. Pan, S.-M. Chen and M.-F. Horng. Vol. 8481. Lecture Notes in Computer Science. Kaohsiung, Taiwan: Springer International Publishing, pp. 440–449. ISBN: 978-3-319-07455-9. DOI: `10.1007/978-3-319-07455-9_46`
2. A. Toubman, J. J. Roessingh, P. Spronck, A. Plaat and H. J. Van den Herik (2014b). Centralized Versus Decentralized Team Coordination Using Dynamic Scripting. In: *Proceedings of the 28th European Simulation and Modelling Conference - ESM'2014*. Ed. by A. C. Brito, J. M. R. Tavares and C. Braganca de Oliveira. Porto, Portugal: Eurosis, pp. 129–134
3. A. Toubman, J. J. Roessingh, P. Spronck, A. Plaat and H. J. Van den Herik (2015a). Rewarding Air Combat Behavior in Training Simulations. In: *Systems, Man, and Cybernetics (SMC), 2015 IEEE International Conference on*. Hong Kong: IEEE Press, pp. 1397–1402. DOI: `10.1109/SMC.2015.248`
4. A. Toubman, J. J. Roessingh, P. Spronck, A. Plaat and H. J. Van den Herik (2015b). Transfer Learning of Air Combat Behavior. In: *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. Miami, Florida: IEEE Press, pp. 226–231. DOI: `10.1109/ICMLA.2015.61`
5. A. Toubman, J. J. Roessingh, J. Van Oijen, M. Hou, L. Luotsinen, J. Harris, R. A. Løvlid, C. Meyer, R. Rijken and M. Turčaník (2016a). Modeling Behavior of Computer Generated Forces with Machine Learning Techniques, the NATO task group approach. In: *Systems, Man, and Cybernetics (SMC), 2016 IEEE International Conference on*. Budapest, Hungary: IEEE. DOI: `10.1109/SMC.2016.7844517`
6. A. Toubman, J. J. Roessingh, P. Spronck, A. Plaat and H. J. Van den Herik (2016b). Rapid Adaptation of Air Combat Behaviour. In: *ECAI 2016 - 22nd European Conference on Artificial Intelligence*. Ed. by G. A. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum and F. Van Harmelen. Vol. 285. Frontiers in Artificial Intelligence and Applications. The Hague, The Netherlands: IOS Press, pp. 1791–1796. DOI: `10.3233/978-1-61499-672-9-1791`
7. A. Toubman (2019). Validating Air Combat Behaviour Models for Adaptive Training of Teams. In: *Adaptive Instructional Systems*. Ed. by R. A. Sottilare and J. Schwarz. Springer International Publishing, pp. 557–571. DOI: `10.1007/978-3-030-22341-0_44`

Curriculum vitae

Armon Toubman was born on October 8, 1988 in Amsterdam, the Netherlands. He attended the Zaanlands Lyceum in Zaandam, where he received his gymnasium diploma in 2006. His love for science fiction, computer programming, and video games led him to start his study of Artificial Intelligence in the same year, at the VU in Amsterdam. He received his bachelor's degree in 2009. During an internship at the Netherlands Organisation for Applied Scientific Research (TNO), he completed his master's thesis on the subject of adaptive autonomy in unmanned ground vehicles. In 2012 he received his master's degree with a specialisation in computational intelligence and self-organisation.

The following year, he joined a collaboration between the Netherlands Aerospace Centre (NLR) and Tilburg University as a PhD candidate. During his research, Armon was supervised by Prof. dr. H.J. van den Herik, Prof. dr. ir. P.H.M. Spronck, Prof. dr. A. Plaat (Tilburg University), and dr. J.J.M. Roessingh (NLR, daily supervision). It is at the NLR that Armon discovered his fourth love, aviation. Midway through his research, he joined professor Van den Herik in his move to establish the Leiden Centre of Data Science (LCDS) at Leiden University. Armon's research has been published at international refereed conferences. At the NLR, he co-supervised four MSc theses together with dr. Roessingh.

Currently, Armon resides in Almere. He works as an R&D engineer at NLR, with a focus on the development of behaviour modelling techniques and machine learning applications. On the web, he can be found at `http://www.armontoubman.com`.

Acknowledgements

Writing a PhD thesis is both the most solitary and most collaborative activity I have undertaken so far. I am most grateful to my supervisor Jaap van den Herik for his guidance, reviews, and endless commitment; to Jan Joris Roessingh, for his daily supervision and the enthusiasm by which he helped to solve a continuous stream of puzzles; and to Pieter Spronck, for sharing not only his dynamic scripting algorithm but also his knowledge and experience of everything surrounding it. Here, I also thank Aske Plaat for his support in the early stages of the research.

Most of my time was spent at the department of Training, Simulation, and Operator Performance at NLR. My sincere gratitude goes to the head of the department, Harrie Bohnen, for providing me with the resources that I needed to complete my research. Many thanks also go to the NLR colleagues who graciously shared their expertise with me.

Reaching outside of my own workplaces, I wish to extend my gratitude to the Royal Netherlands Air Force, and in particular to Rich and the 312, 313, and 322 squadrons. I sincerely hope that your support of my research will double the expected pay back in the near future. Also, in my research, only a few things have been more inspiring than preparing my experiments while listening to the roars of F-16s taking off into the skies. Furthermore, I thank the members of the NATO IST-121 panel for the many interesting discussions we have had, and for helping me place my research in an international context.

My roommate Esther Kuindersma deserves special mention. For me, it was invaluable to share the fantastic ups and the devastating downs that come while writing a thesis, with someone going through the same roller coaster. Thanks, Esther.

Finally, this thesis would not have been possible without the unconditional support of the people closest to me: pa, ma, Omri, Tanna, Iet, Hans, and most of all, Marit. Thank you all.

SIKS dissertation series

2011

01 Botond Cseke (RUN), Variational Algorithms for Bayesian Inference in Latent Gaussian Models

02 Nick Tinnemeier (UU), Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language

03 Jan Martijn van der Werf (TU/e), Compositional Design and Verification of Component-Based Information Systems

04 Hado van Hasselt (UU), Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference

05 Bas van der Raadt (VU), Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.

06 Yiwen Wang (TU/e), Semantically-Enhanced Recommendations in Cultural Heritage

07 Yujia Cao (UT), Multimodal Information Presentation for High Load Human Computer Interaction

08 Nieske Vergunst (UU), BDI-based Generation of Robust Task-Oriented Dialogues

09 Tim de Jong (OU), Contextualised Mobile Media for Learning

10 Bart Bogaert (TiU), Cloud Content Contention

11 Dhaval Vyas (UT), Designing for Awareness: An Experience-focused HCI Perspective

12 Carmen Bratosin (TU/e), Grid Architecture for Distributed Process Mining

13 Xiaoyu Mao (TiU), Airport under Control. Multiagent Scheduling for Airport Ground Handling

14 Milan Lovric (EUR), Behavioral Finance and Agent-Based Artificial Markets

15 Marijn Koolen (UvA), The Meaning of Structure: the Value of Link Evidence for Information Retrieval

Acronyms: SIKS – Dutch Research School for Information and Knowledge Systems; CWI – Centrum voor Wiskunde en Informatica, Amsterdam; EUR – Erasmus Universiteit, Rotterdam; OU – Open Universiteit Nederland; RUG – Rijksuniversiteit Groningen; RUN – Radboud Universiteit Nijmegen; TiU - Tilburg University; TUD – Technische Universiteit Delft; TU/e – Technische Universiteit Eindhoven; UL – Universiteit Leiden; UM – Universiteit Maastricht; UT – Universiteit Twente; UU – Universiteit Utrecht; UvA – Universiteit van Amsterdam; VU – Vrije Universiteit, Amsterdam.

16 Maarten Schadd (UM), Selective Search in Games of Different Complexity

17 Jiyin He (UvA), Exploring Topic Structure: Coherence, Diversity and Relatedness

18 Mark Ponsen (UM), Strategic Decision-Making in complex games

19 Ellen Rusman (OU), The Mind's Eye on Personal Profiles

20 Qing Gu (VU), Guiding service-oriented software engineering - A view-based approach

21 Linda Terlouw (TUD), Modularization and Specification of Service-Oriented Systems

22 Junte Zhang (UvA), System Evaluation of Archival Description and Access

23 Wouter Weerkamp (UvA), Finding People and their Utterances in Social Media

24 Herwin van Welbergen (UT), Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior

25 Syed Waqar ul Qounain Jaffry (VU), Analysis and Validation of Models for Trust Dynamics

26 Matthijs Aart Pontier (VU), Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots

27 Aniel Bhulai (VU), Dynamic website optimization through autonomous management of design patterns

28 Rianne Kaptein (UvA), Effective Focused Retrieval by Exploiting Query Context and Document Structure

29 Faisal Kamiran (TU/e), Discrimination-aware Classification

30 Egon van den Broek (UT), Affective Signal Processing (ASP): Unraveling the mystery of emotions

31 Ludo Waltman (EUR), Computational and Game-Theoretic Approaches for Modeling Bounded Rationality

32 Nees-Jan van Eck (EUR), Methodological Advances in Bibliometric Mapping of Science

33 Tom van der Weide (UU), Arguing to Motivate Decisions

34 Paolo Turrini (UU), Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations

35 Maaike Harbers (UU), Explaining Agent Behavior in Virtual Training

36 Erik van der Spek (UU), Experiments in serious game design: a cognitive approach

37 Adriana Burlutiu (RUN), Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference

38 Nyree Lemmens (UM), Bee-inspired Distributed Optimization

39 Joost Westra (UU), Organizing Adaptation using Agents in Serious Games

40 Viktor Clerc (VU), Architectural Knowledge Management in Global Software Development

41 Luan Ibraimi (UT), Cryptographically Enforced Distributed Data Access Control

42 Michal Sindlar (UU), Explaining Behavior through Mental State Attribution

43 Henk van der Schuur (UU), Process Improvement through Software Operation Knowledge

44 Boris Reuderink (UT), Robust Brain-Computer Interfaces

45 Herman Stehouwer (TiU), Statistical Language Models for Alternative Sequence Selection

46 Beibei Hu (TUD), Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work

47 Azizi Bin Ab Aziz (VU), Exploring Computational Models for Intelligent Support of Persons with Depression

48 Mark Ter Maat (UT), Response Selection and Turn-taking for a Sensitive Artificial Listening Agent

49 Andreea Niculescu (UT), Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality

2012

01 Terry Kakeeto (TiU), Relationship Marketing for SMEs in Uganda

02 Muhammad Umair (VU), Adaptivity, emotion, and Rationality in Human and Ambient Agent Models

03 Adam Vanya (VU), Supporting Architecture Evolution by Mining Software Repositories

04 Jurriaan Souer (UU), Development of Content Management System-based Web Applications

05 Marijn Plomp (UU), Maturing Interorganisational Information Systems

06 Wolfgang Reinhardt (OU), Awareness Support for Knowledge Workers in Research Networks

07 Rianne van Lambalgen (VU), When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions

08 Gerben de Vries (UvA), Kernel Methods for Vessel Trajectories

09 Ricardo Neisse (UT), Trust and Privacy Management Support for Context-Aware Service Platforms

10 David Smits (TU/e), Towards a Generic Distributed Adaptive Hypermedia Environment

11 J.C.B. Rantham Prabhakara (TU/e), Process Mining in the Large: Preprocessing, Discovery, and Diagnostics

12 Kees van der Sluijs (TU/e), Model Driven Design and Data Integration in Semantic Web Information Systems

13 Suleman Shahid (TiU), Fun and Face: Exploring non-verbal expressions of emotion during playful interactions

14 Evgeny Knutov (TU/e), Generic Adaptation Framework for Unifying Adaptive Web-based Systems

15 Natalie van der Wal (VU), Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes.

16 Fiemke Both (VU), Helping people by understanding them - Ambient Agents supporting task execution and depression treatment

17 Amal Elgammal (TiU), Towards a Comprehensive Framework for Business Process Compliance

18 Eltjo Poort (VU), Improving Solution Architecting Practices

19 Helen Schonenberg (TU/e), What's Next? Operational Support for Business Process Execution

20 Ali Bahramisharif (RUN), Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing

21 Roberto Cornacchia (TUD), Querying Sparse Matrices for Information Retrieval

22 Thijs Vis (TiU), Intelligence, politie en veiligheidsdienst: verenigbare grootheden?

23 Christian Muehl (UT), Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction

24 Laurens van der Werff (UT), Evaluation of Noisy Transcripts for Spoken Document Retrieval

25 Silja Eckartz (UT), Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application

26 Emile de Maat (UvA), Making Sense of Legal Text

27 Hayrettin Gurkok (UT), Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games

28 Nancy Pascall (TiU), Engendering Technology Empowering Women

29 Almer Tigelaar (UT), Peer-to-Peer Information Retrieval

30 Alina Pommeranz (TUD), Designing Human-Centered Systems for Reflective Decision Making

31 Emily Bagarukayo (RUN), A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure

32 Wietske Visser (TUD), Qualitative multi-criteria preference representation and reasoning

33 Rory Sie (OU), Coalitions in Cooperation Networks (COCOON)

34 Pavol Jancura (RUN), Evolutionary analysis in PPI networks and applications

35 Evert Haasdijk (VU), Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics

36 Denis Ssebugwawo (RUN), Analysis and Evaluation of Collaborative Modeling Processes

37 Agnes Nakakawa (RUN), A Collaboration Process for Enterprise Architecture Creation

38 Selmar Smit (VU), Parameter Tuning and Scientific Testing in Evolutionary Algorithms

39 Hassan Fatemi (UT), Risk-aware design of value and coordination networks

40 Agus Gunawan (TiU), Information Access for SMEs in Indonesia

41 Sebastian Kelle (OU), Game Design Patterns for Learning

42 Dominique Verpoorten (OU), Reflection Amplifiers in self-regulated Learning

43 Withdrawn

44 Anna Tordai (VU), On Combining Alignment Techniques

45 Benedikt Kratz (TiU), A Model and Language for Business-aware Transactions

46 Simon Carter (UvA), Exploration and Exploitation of Multilingual Data for Statistical Machine Translation

47 Manos Tsagkias (UvA), Mining Social Media: Tracking Content and Predicting Behavior

48 Jorn Bakker (TU/e), Handling Abrupt Changes in Evolving Time-series Data

49 Michael Kaisers (UM), Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions

50 Steven van Kervel (TUD), Ontologogy driven Enterprise Information Systems Engineering

51 Jeroen de Jong (TUD), Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching

2013

01 Viorel Milea (EUR), News Analytics for Financial Decision Support

02 Erietta Liarou (CWI), MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing

03 Szymon Klarman (VU), Reasoning with Contexts in Description Logics

04 Chetan Yadati (TUD), Coordinating autonomous planning and scheduling

05 Dulce Pumareja (UT), Groupware Requirements Evolutions Patterns

06 Romulo Goncalves (CWI), The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience

07 Giel van Lankveld (TiU), Quantifying Individual Player Differences

08 Robbert-Jan Merk (VU), Making enemies: cognitive modeling for opponent agents in fighter pilot simulators

09 Fabio Gori (RUN), Metagenomic Data Analysis: Computational Methods and Applications

10 Jeewanie Jayasinghe Arachchige (TiU), A Unified Modeling Framework for Service Design.

11 Evangelos Pournaras (TUD), Multi-level Reconfigurable Self-organization in Overlay Services

12 Marian Razavian (VU), Knowledge-driven Migration to Services

13 Mohammad Safiri (UT), Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly

14 Jafar Tanha (UvA), Ensemble Approaches to Semi-Supervised Learning

15 Daniel Hennes (UM), Multiagent Learning - Dynamic Games and Applications

16 Eric Kok (UU), Exploring the practical benefits of argumentation in multi-agent deliberation

17 Koen Kok (VU), The PowerMatcher: Smart Coordination for the Smart Electricity Grid

18 Jeroen Janssens (TiU), Outlier Selection and One-Class Classification

19 Renze Steenhuizen (TUD), Coordinated Multi-Agent Planning and Scheduling

20 Katja Hofmann (UvA), Fast and Reliable Online Learning to Rank for Information Retrieval

21 Sander Wubben (TiU), Text-to-text generation by monolingual machine translation

22 Tom Claassen (RUN), Causal Discovery and Logic

23 Patricio de Alencar Silva (TiU), Value Activity Monitoring

24 Haitham Bou Ammar (UM), Automated Transfer in Reinforcement Learning

25 Agnieszka Anna Latoszek-Berendsen (UM), Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System

26 Alireza Zarghami (UT), Architectural Support for Dynamic Homecare Service Provisioning

27 Mohammad Huq (UT), Inference-based Framework Managing Data Provenance

28 Frans van der Sluis (UT), When Complexity becomes Interesting: An Inquiry into the Information eXperience

29 Iwan de Kok (UT), Listening Heads

30 Joyce Nakatumba (TU/e), Resource-Aware Business Process Management: Analysis and Support

31 Dinh Khoa Nguyen (TiU), Blueprint Model and Language for Engineering Cloud Applications

32 Kamakshi Rajagopal (OU), Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development

33 Qi Gao (TUD), User Modeling and Personalization in the Microblogging Sphere

34 Kien Tjin-Kam-Jet (UT), Distributed Deep Web Search

35 Abdallah El Ali (UvA), Minimal Mobile Human Computer Interaction

36 Than Lam Hoang (TU/e), Pattern Mining in Data Streams

37 Dirk Börner (OU), Ambient Learning Displays

38 Eelco den Heijer (VU), Autonomous Evolutionary Art

39 Joop de Jong (TUD), A Method for Enterprise Ontology based Design of Enterprise Information Systems

40 Pim Nijssen (UM), Monte-Carlo Tree Search for Multi-Player Games

41 Jochem Liem (UvA), Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning

42 Léon Planken (TUD), Algorithms for Simple Temporal Reasoning

43 Marc Bron (UvA), Exploration and Contextualization through Interaction and Concepts

2014

01 Nicola Barile (UU), Studies in Learning Monotone Models from Data

02 Fiona Tuliyano (RUN), Combining System Dynamics with a Domain Modeling Method

03 Sergio Raul Duarte Torres (UT), Information Retrieval for Children: Search Behavior and Solutions

04 Hanna Jochmann-Mannak (UT), Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation

05 Jurriaan van Reijsen (UU), Knowledge Perspectives on Advancing Dynamic Capability

06 Damian Tamburri (VU), Supporting Networked Software Development

07 Arya Adriansyah (TU/e), Aligning Observed and Modeled Behavior

08 Samur Araujo (TUD), Data Integration over Distributed and Heterogeneous Data Endpoints

09 Philip Jackson (TiU), Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language

10 Ivan Salvador Razo Zapata (VU), Service Value Networks

11 Janneke van der Zwaan (TUD), An Empathic Virtual Buddy for Social Support

12 Willem van Willigen (VU), Look Ma, No Hands: Aspects of Autonomous Vehicle Control

13 Arlette van Wissen (VU), Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains

14 Yangyang Shi (TUD), Language Models With Meta-information

15 Natalya Mogles (VU), Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare

16 Krystyna Milian (VU), Supporting trial recruitment and design by automatically interpreting eligibility criteria

17 Kathrin Dentler (VU), Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability

18 Mattijs Ghijsen (UvA), Methods and Models for the Design and Study of Dynamic Agent Organizations

19 Vinicius Ramos (TU/e), Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support

20 Mena Habib (UT), Named Entity Extraction and Disambiguation for Informal Text: The Missing Link

21 Kassidy Clark (TUD), Negotiation and Monitoring in Open Environments

22 Marieke Peeters (UU), Personalized Educational Games - Developing agent-supported scenario-based training

23 Eleftherios Sidirourgos (UvA/CWI), Space Efficient Indexes for the Big Data Era

24 Davide Ceolin (VU), Trusting Semi-structured Web Data

25 Martijn Lappenschaar (RUN), New network models for the analysis of disease interaction

26 Tim Baarslag (TUD), What to Bid and When to Stop

27 Rui Jorge Almeida (EUR), Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty

28 Anna Chmielowiec (VU), Decentralized k-Clique Matching

29 Jaap Kabbedijk (UU), Variability in Multi-Tenant Enterprise Software

30 Peter de Cock (TiU), Anticipating Criminal Behaviour

31 Leo van Moergestel (UU), Agent Technology in Agile Multiparallel Manufacturing and Product Support

32 Naser Ayat (UvA), On Entity Resolution in Probabilistic Data

33 Tesfa Tegegne (RUN), Service Discovery in eHealth

34 Christina Manteli (VU), The Effect of Governance in Global Software Development: Analyzing Transactive Memory Systems.

35 Joost van Ooijen (UU), Cognitive Agents in Virtual Worlds: A Middleware Design Approach

36 Joos Buijs (TU/e), Flexible Evolutionary Algorithms for Mining Structured Process Models

37 Maral Dadvar (UT), Experts and Machines United Against Cyberbullying

38 Danny Plass-Oude Bos (UT), Making brain-computer interfaces better: improving usability through post-processing.

39 Jasmina Maric (TiU), Web Communities, Immigration, and Social Capital

40 Walter Omona (RUN), A Framework for Knowledge Management Using ICT in Higher Education

41 Frederic Hogenboom (EUR), Automated Detection of Financial Events in News Text

42 Carsten Eijckhof (CWI/TUD), Contextual Multidimensional Relevance Models

43 Kevin Vlaanderen (UU), Supporting Process Improvement using Method Increments

44 Paulien Meesters (TiU), Intelligent Blauw. Met als ondertitel: Intelligence-gestuurde politiezorg in gebiedsgebonden eenheden.

45 Birgit Schmitz (OU), Mobile Games for Learning: A Pattern-Based Approach

46 Ke Tao (TUD), Social Web Data Analytics: Relevance, Redundancy, Diversity

47 Shangsong Liang (UvA), Fusion and Diversification in Information Retrieval

2015

01 Niels Netten (UvA), Machine Learning for Relevance of Information in Crisis Response

02 Faiza Bukhsh (TiU), Smart auditing: Innovative Compliance Checking in Customs Controls

03 Twan van Laarhoven (RUN), Machine learning for network data

04 Howard Spoelstra (OU), Collaborations in Open Learning Environments

05 Christoph Bösch (UT), Cryptographically Enforced Search Pattern Hiding

06 Farideh Heidari (TUD), Business Process Quality Computation - Computing Non-Functional Requirements to Improve Business Processes

07 Maria-Hendrike Peetz (UvA), Time-Aware Online Reputation Analysis

08 Jie Jiang (TUD), Organizational Compliance: An agent-based model for designing and evaluating organizational interactions

09 Randy Klaassen (UT), HCI Perspectives on Behavior Change Support Systems

10 Henry Hermans (OU), OpenU: design of an integrated system to support lifelong learning

11 Yongming Luo (TU/e), Designing algorithms for big graph datasets: A study of computing bisimulation and joins

12 Julie M. Birkholz (VU), Modi Operandi of Social Network Dynamics: The Effect of Context on Scientific Collaboration Networks

13 Giuseppe Procaccianti (VU), Energy-Efficient Software

14 Bart van Straalen (UT), A cognitive approach to modeling bad news conversations

15 Klaas Andries de Graaf (VU), Ontology-based Software Architecture Documentation

16 Changyun Wei (UT), Cognitive Coordination for Cooperative Multi-Robot Teamwork

17 André van Cleeff (UT), Physical and Digital Security Mechanisms: Properties, Combinations and Trade-offs

18 Holger Pirk (CWI), Waste Not, Want Not! - Managing Relational Data in Asymmetric Memories

19 Bernardo Tabuenca (OU), Ubiquitous Technology for Lifelong Learners

20 Lois Vanhée (UU), Using Culture and Values to Support Flexible Coordination

21 Sibren Fetter (OU), Using Peer-Support to Expand and Stabilize Online Learning

22 Zhemin Zhu (UT), Co-occurrence Rate Networks

23 Luit Gazendam (VU), Cataloguer Support in Cultural Heritage

24 Richard Berendsen (UvA), Finding People, Papers, and Posts: Vertical Search Algorithms and Evaluation

25 Steven Woudenberg (UU), Bayesian Tools for Early Disease Detection

26 Alexander Hogenboom (EUR), Sentiment Analysis of Text Guided by Semantics and Structure

27 Sándor Héman (CWI), Updating compressed colomn stores

28 Janet Bagorogoza (TiU), Knowledge Management and High Performance; The Uganda Financial Institutions Model for HPO

29 Hendrik Baier (UM), Monte-Carlo Tree Search Enhancements for One-Player and Two-Player Domains

30 Kiavash Bahreini (OU), Real-time Multimodal Emotion Recognition in E-Learning

31 Yakup Koç (TUD), On the robustness of Power Grids

32 Jerome Gard (UL), Corporate Venture Management in SMEs

33 Frederik Schadd (TUD), Ontology Mapping with Auxiliary Resources

34 Victor de Graaf (UT), Gesocial Recommender Systems

35 Jungxao Xu (TUD), Affective Body Language of Humanoid Robots: Perception and Effects in Human Robot Interaction

2016

01 Syed Saiden Abbas (RUN), Recognition of Shapes by Humans and Machines

02 Michiel Christiaan Meulendijk (UU), Optimizing medication reviews through decision support: prescribing a better pill to swallow

03 Maya Sappelli (RUN), Knowledge Work in Context: User Centered Knowledge Worker Support

04 Laurens Rietveld (VU), Publishing and Consuming Linked Data

05 Evgeny Sherkhonov (UvA), Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers

06 Michel Wilson (TUD), Robust scheduling in an uncertain environment

07 Jeroen de Man (VU), Measuring and modeling negative emotions for virtual training

08 Matje van de Camp (TiU), A Link to the Past: Constructing Historical Social Networks from Unstructured Data

09 Archana Nottamkandath (VU), Trusting Crowdsourced Information on Cultural Artefacts

10 George Karafotias (VU), Parameter Control for Evolutionary Algorithms

11 Anne Schuth (UvA), Search Engines that Learn from Their Users

12 Max Knobbout (UU), Logics for Modelling and Verifying Normative Multi-Agent Systems

13 Nana Baah Gyan (VU), The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach

14 Ravi Khadka (UU), Revisiting Legacy Software System Modernization

15 Steffen Michels (RUN), Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments

16 Guangliang Li (UvA), Socially Intelligent Autonomous Agents that Learn from Human Reward

17 Berend Weel (VU), Towards Embodied Evolution of Robot Organisms

18 Albert Meroño Peñuela (VU), Refining Statistical Data on the Web

19 Julia Efremova (TU/e), Mining Social Structures from Genealogical Data

20 Daan Odijk (UvA), Context & Semantics in News & Web Search

21 Alejandro Moreno Célleri (UT), From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground

22 Grace Lewis (VU), Software Architecture Strategies for Cyber-Foraging Systems

23 Fei Cai (UvA), Query Auto Completion in Information Retrieval

24 Brend Wanders (UT), Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach

25 Julia Kiseleva (TU/e), Using Contextual Information to Understand Searching and Browsing Behavior

26 Dilhan Thilakarathne (VU), In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains

27 Wen Li (TUD), Understanding Geospatial Information on Social Media

28 Mingxin Zhang (TUD), Large-scale Agent-based Social Simulation - A study on epidemic prediction and control

29 Nicolas Höning (TUD), Peak reduction in decentralised electricity systems - Markets and prices for flexible planning

30 Ruud Mattheij (TiU), The Eyes Have It

31 Mohammad Khelghati (UT), Deep web content monitoring

32 Eelco Vriezekolk (UT), Assessing Telecommunication Service Availability Risks for Crisis Organisations

33 Peter Bloem (UvA), Single Sample Statistics, exercises in learning from just one example

34 Dennis Schunselaar (TU/e), Configurable Process Trees: Elicitation, Analysis, and Enactment

35 Zhaochun Ren (UvA), Monitoring Social Media: Summarization, Classification and Recommendation

36 Daphne Karreman (UT), Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies

37 Giovanni Sileno (UvA), Aligning Law and Action - a conceptual and computational inquiry

38 Andrea Minuto (UT), Materials that Matter - Smart Materials meet Art & Interaction Design

39 Merijn Bruijnes (UT), Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect

40 Christian Detweiler (TUD), Accounting for Values in Design

41 Thomas King (TUD), Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance

42 Spyros Martzoukos (UvA), Combinatorial and Compositional Aspects of Bilingual Aligned Corpora

43 Saskia Koldijk (RUN), Context-Aware Support for Stress Self-Management: From Theory to Practice

44 Thibault Sellam (UvA), Automatic Assistants for Database Exploration

45 Bram van de Laar (UT), Experiencing Brain-Computer Interface Control

46 Jorge Gallego Perez (UT), Robots to Make you Happy

47 Christina Weber (UL), Real-time foresight - Preparedness for dynamic innovation networks

48 Tanja Buttler (TUD), Collecting Lessons Learned

49 Gleb Polevoy (TUD), Participation and Interaction in Projects. A Game-Theoretic Analysis

50 Yan Wang (TiU), The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains

2017

01 Jan-Jaap Oerlemans (UL), Investigating Cybercrime

02 Sjoerd Timmer (UU), Designing and Understanding Forensic Bayesian Networks using Argumentation

03 Daniël Harold Telgen (UU), Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines

04 Mrunal Gawade (CWI), Multi-core Parallelism in a Column-store

05 Mahdieh Shadi (UvA), Collaboration Behavior

06 Damir Vandic (EUR), Intelligent Information Systems for Web Product Search

07 Roel Bertens (UU), Insight in Information: from Abstract to Anomaly

08 Rob Konijn (VU), Detecting Interesting Differences:Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery

09 Dong Nguyen (UT), Text as Social and Cultural Data: A Computational Perspective on Variation in Text

10 Robby van Delden (UT), (Steering) Interactive Play Behavior

11 Florian Kunneman (RUN), Modelling patterns of time and emotion in Twitter #anticipointment

12 Sander Leemans (TU/e), Robust Process Mining with Guarantees

13 Gijs Huisman (UT), Social Touch Technology - Extending the reach of social touch through haptic technology

14 Shoshannah Tekofsky (TiU), You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior

15 Peter Berck (RUN), Memory-Based Text Correction

16 Aleksandr Chuklin (UvA), Understanding and Modeling Users of Modern Search Engines

17 Daniel Dimov (UL), Crowdsourced Online Dispute Resolution

18 Ridho Reinanda (UvA), Entity Associations for Search

19 Jeroen Vuurens (UT), Proximity of Terms, Texts and Semantic Vectors in Information Retrieval

20 Mohammadbashir Sedighi (TUD), Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility

21 Jeroen Linssen (UT), Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)

22 Sara Magliacane (VU), Logics for causal inference under uncertainty

23 David Graus (UvA), Entities of Interest — Discovery in Digital Traces

24 Chang Wang (TUD), Use of Affordances for Efficient Robot Learning

25 Veruska Zamborlini (VU), Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search

26 Merel Jung (UT), Socially intelligent robots that understand and respond to human touch

27 Michiel Joosse (UT), Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors

28 John Klein (VU), Architecture Practices for Complex Contexts

29 Adel Alhuraibi (TiU), From IT-BusinessStrategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT"

30 Wilma Latuny (TiU), The Power of Facial Expressions

31 Ben Ruijl (UL), Advances in computational methods for QFT calculations

32 Thaer Samar (RUN), Access to and Retrievability of Content in Web Archives

33 Brigit van Loggem (OU), Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity

34 Maren Scheffel (OU), The Evaluation Framework for Learning Analytics

35 Martine de Vos (VU), Interpreting natural science spreadsheets

36 Yuanhao Guo (UL), Shape Analysis for Phenotype Characterisation from High-throughput Imaging

37 Alejandro Montes Garcia (TU/e), WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy

38 Alex Kayal (TUD), Normative Social Applications

39 Sara Ahmadi (RUN), Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR

40 Altaf Hussain Abro (VU), Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems

41 Adnan Manzoor (VU), Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle

42 Elena Sokolova (RUN), Causal discovery from mixed and missing data with applications on ADHD datasets

43 Maaike de Boer (RUN), Semantic Mapping in Video Retrieval

44 Garm Lucassen (UU), Understanding User Stories - Computational Linguistics in Agile Requirements Engineering

45 Bas Testerink (UU), Decentralized Runtime Norm Enforcement

46 Jan Schneider (OU), Sensor-based Learning Support

47 Jie Yang (TUD), Crowd Knowledge Creation Acceleration

48 Angel Suarez (OU), Collaborative inquiry-based learning

2018

01 Han van der Aa (VU), Comparing and Aligning Process Representations

02 Felix Mannhardt (TU/e), Multi-perspective Process Mining

03 Steven Bosems (UT), Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction

04 Jordan Janeiro (TUD), Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks

05 Hugo Huurdeman (UvA), Supporting the Complex Dynamics of the Information Seeking Process

06 Dan Ionita (UT), Model-Driven Information Security Risk Assessment of Socio-Technical Systems

07 Jieting Luo (UU), A formal account of opportunism in multi-agent systems

08 Rick Smetsers (RUN), Advances in Model Learning for Software Systems

09 Xu Xie (TUD), Data Assimilation in Discrete Event Simulations

10 Julienka Mollee (VU), Moving forward: supporting physical activity behavior change through intelligent technology

11 Mahdi Sargolzaei (UvA), Enabling Framework for Service-oriented Collaborative Networks

12 Xixi Lu (TU/e), Using behavioral context in process mining

13 Seyed Amin Tabatabaei (VU), Computing a Sustainable Future

14 Bart Joosten (TiU), Detecting Social Signals with Spatiotemporal Gabor Filters

15 Naser Davarzani (UM), Biomarker discovery in heart failure

16 Jaebok Kim (UT), Automatic recognition of engagement and emotion in a group of children

17 Jianpeng Zhang (TU/e), On Graph Sample Clustering

18 Henriette Nakad (UL), De Notaris en Private Rechtspraak

19 Minh Duc Pham (VU), Emergent relational schemas for RDF

20 Manxia Liu (RUN), Time and Bayesian Networks

21 Aad Slootmaker (OU), EMERGO: a generic platform for authoring and playing scenario-based serious games

22 Eric Fernandes de Mello Araujo (VU), Contagious: Modeling the Spread of Behaviours, Perceptions and Emotions in Social Networks

23 Kim Schouten (EUR), Semantics-driven Aspect-Based Sentiment Analysis

24 Jered Vroon (UT), Responsive Social Positioning Behaviour for Semi-Autonomous Telepresence Robots

25 Riste Gligorov (VU), Serious Games in Audio-Visual Collections

26 Roelof Anne Jelle de Vries (UT),Theory-Based and Tailor-Made: Motivational Messages for Behavior Change Technology

27 Maikel Leemans (TU/e), Hierarchical Process Mining for Scalable Software Analysis

28 Christian Willemse (UT), Social Touch Technologies: How they feel and how they make you feel

29 Yu Gu (TiU), Emotion Recognition from Mandarin Speech

30 Wouter Beek (VU), The “K” in “semantic web” stands for “knowledge”: scaling semantics to the web

2019

01 Rob van Eijk (UL), Web privacy measurement in real-time bidding systems. A graph-based approach to RTB system classification

02 Emmanuelle Beauxis Aussalet (CWI, UU), Statistics and Visualizations for Assessing Class Size Uncertainty

03 Eduardo Gonzalez Lopez de Murillas (TU/e), Process Mining on Databases: Extracting Event Data from Real Life Data Sources

04 Ridho Rahmadi (RUN), Finding stable causal structures from clinical data

05 Sebastiaan van Zelst (TU/e), Process Mining with Streaming Data

06 Chris Dijkshoorn (VU), Nichesourcing for Improving Access to Linked Cultural Heritage Datasets

07 Soude Fazeli (TUD), Recommender Systems in Social Learning Platforms

08 Frits de Nijs (TUD), Resource-constrained Multi-agent Markov Decision Processes

09 Fahimeh Alizadeh Moghaddam (UvA), Self-adaptation for energy efficiency in software systems

10 Qing Chuan Ye (EUR), Multi-objective Optimization Methods for Allocation and Prediction

11 Yue Zhao (TUD), Learning Analytics Technology to Understand Learner Behavioral Engagement in MOOCs

12 Jacqueline Heinerman (VU), Better Together

13 Guanliang Chen (TUD), MOOC Analytics: Learner Modeling and Content Generation

14 Daniel Davis (TUD), Large-Scale Learning Analytics: Modeling Learner Behavior & Improving Learning Outcomes in Massive Open Online Courses

15 Erwin Walraven (TUD), Planning under Uncertainty in Constrained and Partially Observable Environments

16 Guangming Li (TU/e), Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models

17 Ali Hurriyetoglu (RUN), Extracting actionable information from microtexts

18 Gerard Wagenaar (UU), Artefacts in Agile Team Communication

19 Vincent Koeman (TUD), Tools for Developing Cognitive Agents

20 Chide Groenouwe (UU), Fostering technically augmented human collective intelligence

21 Cong Liu (TU/e), Software Data Analytics: Architectural Model Discovery and Design Pattern Detection

22 Martin van den Berg (VU), Improving IT Decisions with Enterprise Architecture

23 Qin Liu (TUD), Intelligent Control Systems: Learning, Interpreting, Verification

24 Anca Dumitrache (VU), Truth in Disagreement - Crowdsourcing Labeled Data for Natural Language Processing

25 Emiel van Miltenburg (VU), Pragmatic factors in (automatic) image description

26 Prince Singh (UT), An Integration Platform for Synchromodal Transport

27 Alessandra Antonaci (OU), The Gamification Design Process applied to (Massive) Open Online Courses

28 Esther Kuindersma (UL), Cleared for take-off: Game-based learning to prepare airline pilots for critical situations

29 Daniel Formolo (VU), Using virtual agents for simulation and training of social skills in safety-critical circumstances

30 Vahid Yazdanpanah (UT), Multiagent Industrial Symbiosis Systems

31 Milan Jelisavcic (VU), Alive and Kicking: Baby Steps in Robotics

32 Chiara Sironi (UM), Monte-Carlo Tree Search for Artificial General Intelligence in Games

33 Anil Yaman (TU/e), Evolution of Biologically Inspired Learning in Artificial Neural Networks

34 Negar Ahmadi (TU/e), EEG Microstate and Functional Brain Network Features for Classification of Epilepsy and PNES

35 Lisa Facey-Shaw (OU), Gamification with digital badges in learning programming

36 Kevin Ackermans (OU), Designing Video-Enhanced Rubrics to Master Complex Skills

37 Jian Fang (TUD), Database Acceleration on FPGAs

38 Ákos Kádár (TiU), Learning visually grounded and multilingual representations

2020

01 Armon Toubman (UL), Calculated Moves: Generating Air Combat Behaviour