Calculated Moves: Generating Air Combat Behaviour

Toubman, A.

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).

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

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).

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

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).

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

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

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

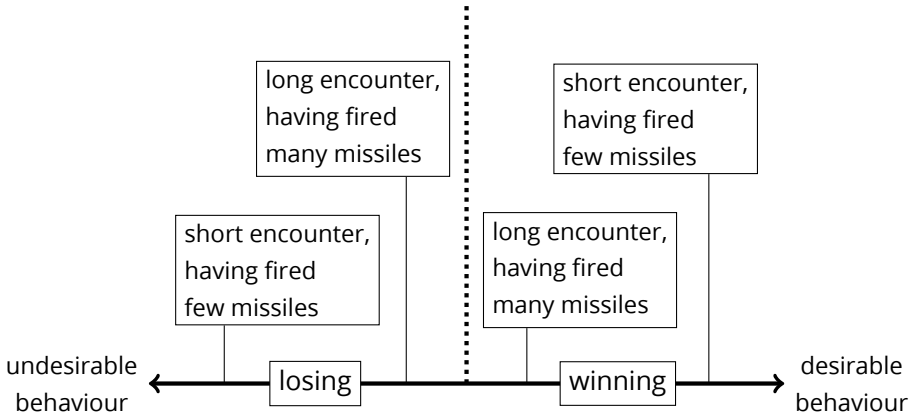

Figure 4.1 The spectrum of desirable air combat behaviour.

possible, viz. $r_{\text{duration}} \to 0$. 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 $r_{\text{missiles}} \to 1$. 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 $r_{\text{missiles}} \to 0$ 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

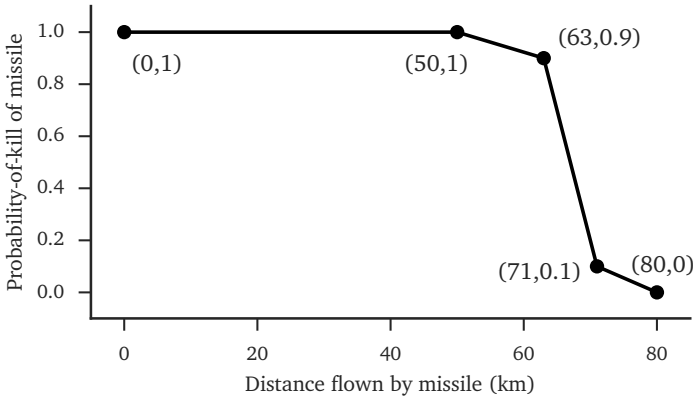

Figure 4.2 The probability-of-kill curve of the missiles in LWACS. The probability-of-kill of missiles depends on the number of kilometres flown by the missiles between (a) the moment of launch and (b) the moment of intercept. We define the curve by the following five points: (0,1), (50,1), (63,0.9),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

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

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).

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

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

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 $r_1 = P_{k1} * m_1$. 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: $r_i = P_{ki} * m_i$, 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 $P_{k1} = 0.1$, 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 $P_{k2} = 0.8$, 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.

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

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

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

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.

$$\text{AA-REWARD}(s_{\text{terminal}}) = \sum_{\{n \,|\, \text{team}(n)\}} r_{\text{intercept}}(n) \tag{4.21}$$

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.

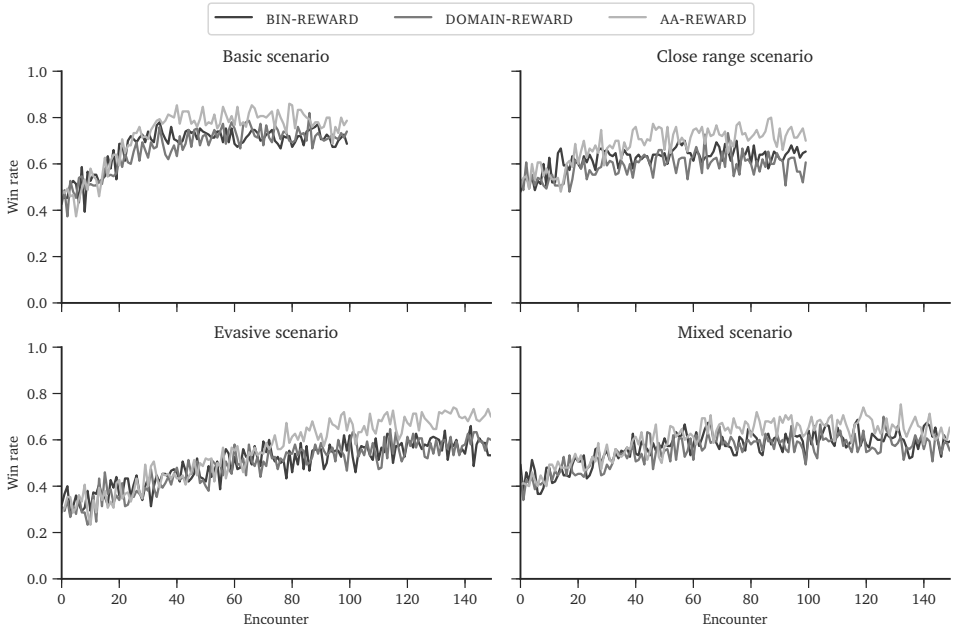

Figure 4.3 The win rates of red against blue. Red used one of three reward functions: (1) BIN-REWARD, (2) DOMAIN-REWARD, and (3) AA-REWARD. The behaviour of blue depended on four scenarios: (1) the basic scenario, (2) the close range scenario, (3) the evasive scenario, and (4) the mixed scenario.

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%.