LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

INSTITUT FÜR INFORMATIK
LEHRSTUHL FÜR MOBILE UND VERTEILTE SYSTEME

MASTERARBEIT

Hidden Attacks in Multi-Agent Reinforcement Learning

Arnold Unterauer

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
LEHRSTUHL FÜR MOBILE UND VERTEILTE SYSTEME

MASTERARBEIT

Hidden Attacks in Multi-Agent Reinforcement Learning

Arnold Unterauer

Aufgabensteller:	Prof. Dr. Claudia Linnhoff-Popien
Betreuer:	Thomy Phan Philipp Altmann
Abgabetermin:	24. Dezember 2022

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 24. Dezember 2022

. . . . . .

(Unterschrift des Kandidaten)

Abstract

Multi-Agent Reinforcement Learning has become more and more important in recent years. Whether in the areas of logistics, robotics or transportation, everywhere the objective is to increase the performance. While this has been the main focus, the question of robustness and resilience of such agents has become increasingly relevant. Adversarial Attacks can often affect agents, resulting in a decrease in performance or even total failure. While many works around Adversarial Attacks have been focused on the perception and/or the environment, the attack vector originating from agents has been less investigated. This may involve agents being affected by safety issues, such as malfunctions or outages. On the other hand, agents may also be manipulated by external malicious intent. In this work we will take a closer look at the security aspect and the resulting impact of such externally compromised agents (attackers) on other actors (protagonists). In this process we introduce Infiltrating Stealth Agent Attack Controller (ISAAC), a new approach which leverages this attack vector to reduce the performance of the protagonists while at the same time remaining hidden with respect to external observers. For this purpose we design our Adversarial Attack to be natural and a black box attack, thus representing a scenario which is as close to real life as possible. Thereby we simulate a malicious attack that does not possess any additional information or knowledge regarding the protagonists. In order to evaluate ISAAC, we will look at various state-of-the-art algorithms and perform attacks on them. Additionally, to measure the success of our approach using a number of different metrics, we will utilize several scenarios from the StarCraft Multi-Agent Challenge (SMAC). In the SMAC environments agents have to cooperate with each other to be successful in a continuous setting. During this process we will observe that for various algorithms ISAAC is able to significantly decrease the performance of the protagonists while remaining hidden. As a result we provide a benchmark tool to study and measure the robustness and resilience of algorithms in multi-agent systems with respect to Adversarial Attacks originating from agents.

Contents

1. Introduction

Artificial Intelligence (AI) has become increasingly more relevant over the past few years. Whether in the areas of Biology [JEP+21], Digital Media [RDN+22, RBL+21, WLS+22], or Natural Language Processing (NLP) [BMR+20], Machine Learning (ML) is widely adopted nowadays. With growing compute power Big Data or Deep Learning operations can be performed progressively faster, resulting at the same time in more and more applications of ML in our daily lives. However, with increasing usage of AI, the question of the safety and especially the security aspects of such ML models is becoming prevalent. These can be extremely important in various applications, as for example in critical infrastructures or in safety-critical situations. Otherwise a defective model may cause severe damage or even put human lives at risk. For this reason the importance of guaranteeing the correct functionality of ML models and providing sufficient level of defense in terms of security has risen. However, while testing and verification are commonly applied to traditional software components, machine learning models tend to be quite challenging to verify [KGT+0]. This is particularly evident when considering that most of the works on machine learning approaches do not address any of the safety and security concerns, but rather focus on overall performance. Consequently often undesirable side effects occur during the deployment of the fully trained models, such as the discrimination against minority groups [PP19, MMS+19] or model inversion [WFL+22]. Despite this challenge, as the number of AI applications continues to grow, so does the amount of research being conducted on the topics of safety and security. Regarding the safety, the reliability and correctness of the models, there have been many works which do not only demonstrate the weaknesses [CW16, MFF15], but also actively attempt to solve these issues [MMS+17, RSL18]. Similar to safety security, the ability of the model to protect itself from malicious or disruptive influences, has seen a lot of progress in recent years. As a result potential security issues have been identified [Kis19, KP19, NMSE20] or defense mechanisms were introduced [IRM+19, SWW+19, GLZ+20] in order to improve the resilience of ML models. Also in the area of Reinforcement Learning, a variety of Adversarial Attacks, inputs that cause incorrect decisions or performance degradation of models, have been widely studied to reveal weaknesses [CLX+19] or to improve the robustness [PTL+17] and resilience [KS17] of the models. One field in which Reinforcement Learning can be applied are agent systems, where actors or agents have to perform specified tasks in a given environment. In this context Adversarial Attacks are typically carried out using the observations of the agents [ZCBH21, TWW+21, XEZJ22] or the environment itself [RRD+20, HZ19]. Although, it is also possible to make use of the agents in an environment as a potential attack vector in order to target multi-agent systems [PGS+20]. Especially when considering cooperative settings, this may lead to severe reductions in performance as a result of sabotage.

In this work we will take a closer look at this rather novel attack vector. We introduce Infiltrating Stealth Agent Attack Controller (ISAAC), a new approach representing a potentially realistic attack scenario. The attack is considered to be natural, iterative,

optimized, intentional, targeted, false positive as well as a black box attack. Our approach substitutes agents in a multi-agent system and thus infiltrates the team of the performers optimizing the original objective, known as protagonists. While the attackers or invaders also sabotage the team, the main focus in this work lies far more on the hiding of such an attack. For this purpose we make use of techniques taken from the field of computer vision [GPAM+14] in order to imitate the behavior of the protagonists. ISAAC thereby succeeds in not only decreasing the performance of the agents in a cooperative multi-agent system, but at the same time in preventing the attackers from being identified by external observers. Additionally we are able to change the priority of each objective in the context of a Multi Objective Optimization Problem [Gun18] and thus provide an appropriate trade-off for the attackers. The primary motivation behind this work is to not only raise more awareness against such malicious and stealthy Adversarial Attacks on multi-agent systems, but also to provide the ISAAC as a type of benchmark for the resilience of Reinforcement Learning algorithms.

Our primary contributions using ISAAC to the field of Reinforcement Learning can be summarized as follows:

- In order to provide a comprehensive overview of ISAAC, we will first introduce the key components that make up its functionality. These consist of two essential parts: an antagonistic one, which is designed to reduce the performance of the protagonists and an imitation component, which allows the invader to be hidden from the perspective of external observers.

- We will subsequently apply ISAAC in experiments on fully trained agents in various scenarios of the SMAC domain to demonstrate the effectiveness of our approach. These experiments will allow us to showcase the success of ISAAC in a diverse range of situations.

- The results of a survey will be evaluated in order to assess the ability of ISAAC Invaders to hide. Due to the absence of a comprehensive metric that meets our requirements, we utilize this survey as a point of reference for hiding attackers.

- Challenges and issues that may emerge due to different types of algorithms as well as environments will also be addressed in regards to ISAAC.

2. Background

2.1. Decentralized partially observable Markov Decision Process

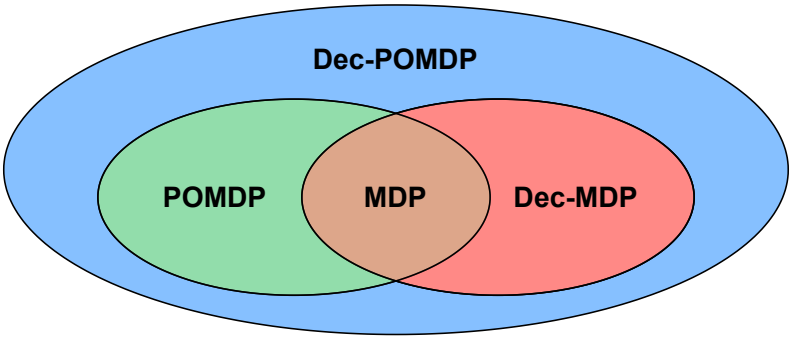


Figure 2.1.: Relationship between different agent system models, adopted from [BZI13].

The problems agents have to face in a multi-agent system are typically modeled as Markov Decision Process (MDP) [Bel57]. If the environment can only be partially observed by the agents, this is referred to as a Partially Observable MDP (POMDP) [JSJ94]. Additionally, if the action selection of the agents is made in a decentralized fashion by a set of decision makers, then the problem has to be further generalized such that one obtains a decentralized POMDP (Dec-POMDP) [Oli12]. A Dec-POMDP is defined by a tuple $M_{\text{Dec-POMDP}} =< \mathcal{S}, \mathcal{U}, \mathcal{P}, r, \mathcal{Z}, \mathcal{O}, n, \gamma, b_0 >$, where a set of finite agents $a \in \mathcal{A} \equiv \{1, .., n\}$ interact with the environment. The environment has $s \in \mathcal{S}$ global true states in the process, however these states can not be viewed by agents directly. Instead each agent obtains an individual partial observation $z_i \in \mathcal{Z}$ from the partially observable setting according to the the observation probability function $\mathcal{O}(z_i|s_t, a_i) : \mathcal{S} \times \mathcal{A} \to [0, 1]$. Using this partial observation, the agents take a joint action $\boldsymbol{u} \in \boldsymbol{\mathcal{U}} \equiv \mathcal{U}^n$, which consists of the actions $u_i \in \mathcal{U}$ resulting from the simultaneous action selection of the agents at each time step. Once a joint action is executed, it leads to a transition in the environment according to the state transition function $\mathcal{P}(s_{t+1}|s_t, \boldsymbol{u}_t) : \mathcal{S} \times \boldsymbol{\mathcal{U}} \times \mathcal{S} \to [0, 1]$. Subsequently the agents receive a team reward R_t from the same shared joint reward function $r(s_t, \boldsymbol{u}_t) : \mathcal{S} \times \boldsymbol{\mathcal{U}} \to \mathbb{R}$. Additionally the initial stochastic state distribution is defined by $b_0 : \mathcal{S} \to [0, 1]$ and the discount factor is given as $\gamma \in [0, 1)$. Apart from the individual partial observations z_i of the agents, they also possess an individual action-observation history $\tau_i \in \mathcal{T} \equiv (\mathcal{Z} \times \mathcal{U})^*$. By considering all agents collectively, the combination of these observations yields an joint action-observation history $\boldsymbol{\tau} \in \boldsymbol{\mathcal{T}} \equiv \mathcal{T}^n$. On the basis of this, agents shape their own individual stochastic policy $\pi_i(u_i|\tau_i) : \mathcal{T} \times \mathcal{U} \to [0, 1]$ to maximize their own performance. We can also represent the policy of all agents as a stochastic joint policy $\boldsymbol{\pi} = \langle \pi_1, ..., \pi_n \rangle$. By using the joint policy $\boldsymbol{\pi}$ and the discounted return $\mathcal{G}_t = \sum_{k=0}^{\infty} \gamma^k r(s_{t+k}, \boldsymbol{u}_{t+k})$, we construct the joint value function $V^{\boldsymbol{\pi}}(s_t) = \mathbb{E}_{\boldsymbol{\pi}}[\mathcal{G}_t|s_t]$. Taking the joint action $\boldsymbol{u}$ into account,

we obtain the joint action-value function $Q^{\boldsymbol{\pi}}(s_t, \boldsymbol{u}_t) = \mathbb{E}_{\boldsymbol{\pi}}[\mathcal{G}_t | s_t, \boldsymbol{u}_t]$ which we intend to maximize.

2.2. Deep Q-Learning

Deep Q-learning [WD92, SB98, MKS+13] is a method used to represent the action-value function Q^{π} with the help of a Deep Neural Network, such as a Deep Q-Network (DQN) [MKS+15], which utilizes iterative training to find the optimal joint action-value function $Q^*(s_t, \boldsymbol{u}_t) = \mathbb{E}_{\boldsymbol{\pi}}[\max_{\boldsymbol{u}_{t+1}} Q^*(s_{t+1}, \boldsymbol{u}_{t+1})]$. In order to accomplish this, we use the experience E that the agents have accumulated during their interaction with the environment. For each time step t and agent i, the experience can thus be defined by a experience tuple $e_{t,i} = \langle s_t, z_{t,i}, u_{t,i}, r_t, s_{t+1}, z_{t+1,i} \rangle \in E$ and combined into a global experience tuple $\boldsymbol{e}_t = \langle s_t, z_t, \boldsymbol{u}_t, r_t, s_{t+1}, z_{t+1} \rangle \in \boldsymbol{E}$. Here s_{t+1} and z_{t+1} denote the successor state and observation after the joint action $\boldsymbol{u}_t$ was taken in the environment at time step t with state s_t and the observation z_t, which resulted in a reward of r_t. While this is sufficient to update the θ parameterized Deep Neural Network to approximate Q^*, there are some other approaches that additionally include the joint action-observation history $\boldsymbol{\tau}$ at time step t and $t+1$, giving us the global history experience tuple $\boldsymbol{e}^{\boldsymbol{\tau}}_t = \langle s_t, z_t, \boldsymbol{\tau}_t, \boldsymbol{u}_t, r_t, s_{t+1}, z_{t+1}, \boldsymbol{\tau}_{t+1} \rangle \in \boldsymbol{E}^{\boldsymbol{\tau}}$. In order to benefit from these histories [McC95], there are so called Deep Recurrent Q-Networks (DRQN) [HS15] that make use of components from recurrent neural networks. For instance, there are Long Short-Term Memory (LSTM) [HS97] or Gated Recurrent Unit (GRU) [CvMBB14] architectures that are used for learning over an extended time frame. Using the Deep Recurrent Q-Networks and with the help of the global history experiences $\boldsymbol{e}^{\boldsymbol{\tau}}$, it is possible to learn the parameters θ by minimizing the expected squared temporal difference error [Sut88]:

$$\mathcal{L}(\theta) = \mathbb{E}_{\boldsymbol{e}^{\boldsymbol{\tau}}} \left[\left(r + \gamma \max_{\boldsymbol{u}_{t+1}} Q^{\boldsymbol{\pi}}(\boldsymbol{\tau}_{t+1}, \boldsymbol{u}_{t+1}; \theta^-) - Q^{\boldsymbol{\pi}}(\boldsymbol{\tau}_t, \boldsymbol{u}_t; \theta) \right)^2 \right] \tag{2.1}$$

where θ^- are the parameters of a target network which are periodic copied snapshots of θ and held constant over a time period.

Besides the history based architectures of neural networks, a paradigm in cooperative multi agent systems called Centralized Training with Decentralized Execution (CTDE) [OSV11] is widely being used. While the individual agents in a multi-agent system gather their own experience during decentralized execution, they are not able to access additional information. This is where CTDE provides additional resources through centralized training, which allows supplementary information to be incorporated into the learning of each individual agent, such as the experiences of other agents or even the global state of the environment. For this paradigm to be applied, the condition must hold that the joint action-value function $Q^{\boldsymbol{\pi}}_{tot} : \boldsymbol{\mathcal{T}} \times \boldsymbol{\mathcal{U}} \to \mathbb{R}$ can be factorized into the individual action-value functions $[Q^{\pi}_i : \mathcal{T} \times \mathcal{U} \to \mathbb{R}]^n_{i=1}$ of the different agents, also referred as Individual-Global-Max (IGM) [SKK+19]:

$$\underset{\boldsymbol{u} \in \boldsymbol{\mathcal{U}}}{\operatorname{argmax}} Q^{\boldsymbol{\pi}}_{tot}(\boldsymbol{\tau}, \boldsymbol{u}) = \left(\underset{u_1 \in \mathcal{U}^1}{\operatorname{argmax}} Q^{\pi}_1(\tau_1, u_1), ..., \underset{u_n \in \mathcal{U}^n}{\operatorname{argmax}} Q^{\pi}_n(\tau_n, u_n) \right). \tag{2.2}$$

Using this paradigm, the individual agents can perform their action selection during

decentralized execution with respect to their own individual action-value function Q_i^π, which at the same time will also maximize the joint action-value function $Q_{tot}^{\boldsymbol{\pi}}$. Thus the agents are independent of each other during their interaction in the environment and only during training additional information is being utilized. In the following section we will introduce several value-based algorithms that may also utilize previously described architectures and paradigms.

Algorithms

Independent Q-Learning

Perhaps the most simplistic way to view a multi-agent system is to treat each agent as an independent learning actor in a shared environment. Thus it is possible to divide a multi-agent system into multiple single-agent systems. This is the underlying concept of the Independent Q-Learning (IQL) [Tan93] algorithm, where all agents have to solve their own problems independently from other agents. Despite the fairly good performance of IQL, there are some issues which the algorithm does not address. For instance the non-stationarity in the environment, which also includes the policies of all agents, is a crucial part of the optimization and is not taken into account by IQL, resulting in no guarantee of convergence for the training process [LSC21]. Furthermore the goal of the environment may require cooperation between different agents, which might be a difficulty for IQL agents, as they only try to optimize their own policy and not the joint policy.

Value Decomposition Networks

In Value Decomposition Networks (VDN) [SLG$^+$17], different from the IQL, the agents are not viewed as independent learners but as connected ones. Here VDN utilizes the CTDE paradigm by using a decomposition of the action-value function $Q_{tot}^{\boldsymbol{\pi}}$:

$$Q_{tot}^{\boldsymbol{\pi}}(\boldsymbol{\tau}, \boldsymbol{u}) \approx \sum_{i=1}^{n} Q_i^\pi(\tau_i, u_i). \tag{2.3}$$

By this additive factorization the requirement IGM for the CTDE is fulfilled. The centralized training of the agents is executed according to the corresponding updates by 2.1. While this additivity is considerably more restrictive than the minimum requirement of IGM, the approach achieves quite strong performance in a wide number of environments and provides a solid foundation for several subsequent algorithms [RSdW$^+$18].

QMIX

QMIX [RSdW$^+$18] tackles the limitation of the VDN algorithm. Here the problem is also approached in a CTDE fashion, however the additivity is replaced by a less restrictive constraint. Instead of the additivity the authors of QMIX make use of a monotonicity property, which allows them to satisfy the individual argmax arguments defined in IGM by applying the following condition:

$$\frac{\partial Q_{tot}^{\boldsymbol{\pi}}(\boldsymbol{\tau}, \boldsymbol{u})}{\partial Q_i^\pi(\tau_i, u_i)} \geq 0, \quad \forall i \in \mathcal{A} \tag{2.4}$$

Since QMIX is less constrained, it is capable of completing a greater amount of environments with success when compared against VDN, while still meeting the requirements of IGM [RSdW+18].

Weighted QMIX

In WQMIX [RFPW20] the potential challenge is further elaborated in case the IGM requirement is not given. It may be possible that a argmax action-value function $Q_{tot}^{\boldsymbol{\pi}}$ is not factorizable into individual terms, because the argmax action-value functions Q_i^{π} of the individual agents may contradict with the joint one. For this reason WQMIX introduces weights to ensure that the joint action-value function is still maximized. One type of weighting is defined as Optimistic Weighting (OW-QMIX), in which potentially underestimated joint actions are weighted more heavily with respect to $Q^{\boldsymbol{\pi}}$ in the hope that they might be a superior joint action:

$$w(\boldsymbol{\tau}, \boldsymbol{u}) = \begin{cases} 1, & \text{if } Q_{tot}^{\boldsymbol{\pi}}(\boldsymbol{\tau}, \boldsymbol{u}) < Q^{\boldsymbol{\pi}}(\boldsymbol{\tau}, \boldsymbol{u}) \\ \alpha_W, & \text{otherwise} \end{cases}, \quad \exists \alpha_W > 0. \tag{2.5}$$

Despite the fact that WQMIX does not always satisfy the requirement specified by IGM, in practice the approach performs fairly well. However there can be performance drops in more demanding tasks [WRL+20].

QTRAN

The concept of QTRAN [SKK+19] is based on the original QMIX. However, instead of performing the factorization on the action-value function $Q_{tot}^{\boldsymbol{\pi}}$ itself, QTRAN makes use of a separate transformed action-value function $Q_{tot}^{\prime\boldsymbol{\pi}}$. In this way, it is possible to overcome the monotonic limitation of QMIX and thus solve a greater number of problems. For this purpose QTRAN introduces the following linear summation for the transformed action-value function:

$$Q_{tot}^{\prime\boldsymbol{\pi}}(\boldsymbol{\tau}, \boldsymbol{u}) = \sum_{i=1}^{N} Q_i^{\pi}(\tau_i, u_i) + V^t(\boldsymbol{\tau}), \tag{2.6}$$

where $V^t(\boldsymbol{\tau}) = max_u Q_{tot}^{\boldsymbol{\pi}}(\boldsymbol{\tau}, \boldsymbol{u}) - \sum_{i=1}^{N} Q_i^{\pi}(\tau_i, \dot{u}_i)$ is the value estimator representing the gap between the individual action-value function Q_i^{π} sum and the joint action-value function $Q_{tot}^{\boldsymbol{\pi}}$, and $\dot{u}_i$ are the individual optimal local actions. While this transformation loosens the requirements with respect to IGM, in practice the performance of QTRAN has shown mixed results. In some cases the algorithm may perform worse when compared against QMIX [MRSW19].

QPLEX

While QTRAN and QMIX always meet the requirements of IGM, Duplex Dueling Multi-Agent Q-Learning (QPLEX) [WRL+20] attempts to approximately satisfy them. Therefore a dueling architecture is used, consisting of a joint and an individual action-value function. This allows the neural network of QPLEX to learn and approximate a so-called advantage-based IGM:

$$\underset{\boldsymbol{u}\in\boldsymbol{\mathcal{U}}}{\operatorname{argmax}}\,\mathcal{A}_{tot}(\boldsymbol{\tau},\boldsymbol{u}) = \left(\underset{u_1\in\mathcal{U}^1}{\operatorname{argmax}}\,\mathcal{A}_1(\tau_1,u_1),...,\underset{u_n\in\mathcal{U}^n}{\operatorname{argmax}}\,\mathcal{A}_n(\tau_n,u_n)\right), \tag{2.7}$$

with the Joint and Individual Dueling:

$$Q^{\boldsymbol{\pi}}_{tot}(\boldsymbol{\tau},\boldsymbol{u}) = V^p_{tot}(\boldsymbol{\tau}) + A_{tot}(\boldsymbol{\tau},\boldsymbol{u}) \quad , \quad V^p_{tot}(\boldsymbol{\tau}) = \max_{\boldsymbol{u'}} Q^{\boldsymbol{\pi}}_{tot}(\boldsymbol{\tau},\boldsymbol{u'}) \tag{2.8}$$

$$Q^{\pi}_{i}(\tau_i,u_i) = V^p_i(\tau_i) + A_i(\tau_i,u_i) \quad , \quad V^p_i(\tau_i) = \max_{u'_i} Q^{\pi}_i(\tau_i,u'_i). \tag{2.9}$$

As a result of this approximation, QPLEX succeeds in performing rather well when compared to other algorithms.

2.3. Zero-Sum Games

Assuming a system has two opposing parties, we can define such a system as a Zero-Sum Game [Lit94]. Therefore the previously described value function becomes a maximization of the minimization of the following:

$$V_{minmax}(s_t) = \max_{\pi_i}\min_{u_t,j}\sum_{u_t,i}\pi_i(u_{t,i}|\tau_{t,i})Q^{\pi}_i(s_t,u_t), \tag{2.10}$$

where j denotes the agents who pursue the opposing goal of the i agents. Since both sides have exactly the opposite objective, the maximization of the agent's reward becomes the minimization of the other party's one. Thus we only need the action-value function $Q^{\boldsymbol{\pi}}$ of one party to determine both of them: $Q^{\boldsymbol{\pi}}_j = -Q^{\boldsymbol{\pi}}_i$.

2.4. Generative Adversarial Networks

Generative Adversarial Networks (GANs) [GPAM$^+$14] have gained a lot of popularity in the past few years. They are used to learn an underlying distribution of data and generate new similar ones from that distribution. This opens up a wide range of possibilities, from data generation for too small data sets [NH19] to image creation with specific art styles [XLWW21, XWF$^+$18, GEB15], GANs are used in a wide variety of fields. The GANs consist of two main components, which are commonly represented by Deep Neural Networks: the generator and the discriminator. The generator tries to recreate the distribution of the training data as accurately as possible from a given distribution (usually uniformly) [GPAM$^+$14]. Whereas the discriminator serves as a classifier with the task of distinguishing between the fake data created by the generator and the real training data. As such, the two parts pursue different opposing goals. This opposition can also be formulated as a two-player min-max game, where both players are iteratively trained:

$$\min_G\max_D B(G,D) = \min_G\max_D \mathbb{E}_{x\sim p_x}[\log D(x)] + \mathbb{E}_{z\sim p_d}[\log(1-D(G(z)))] \tag{2.11}$$

Both parties try to mini- or maximize this respective value function $B(G,D)$. Here,

the value function $B(G, D)$ is usually a function that measures the difference between two distributions. Hence the interest of the generator G is to minimize this value function where it samples data m from a distribution M and maps it into the distribution X of the training data. This creates an implicit probability distribution p_d, which through continuous training converges to the probability distribution p_x of the real training data. In contrast, the discriminator D tries to maximize the value function $B(G, D)$. For this purpose, the discriminator takes samples x from the entire data pool and classifies them into the two categories: True (1), if the data belongs to the real training data or False (0), if the generator created this sample.

In other words, the objective of the generator is to fool the discriminator by creating samples that are as indistinguishable as possible from the real training data. Whereas the discriminator has to detect the samples created by the generator and correctly identify them as fakes. In order to train both components, we first use the generator to produce samples and then make use of the discriminator to label these samples and the real training data. Afterwards, with the help of the value function, we can measure the difference between the underlying distributions. For the value function $B(G, D)$ the Binary Cross Entropy (BCE) [Rub99] is commonly chosen:

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))] \tag{2.12}$$

Here $H_p(q)$ stands for Binary Cross Entropy, which is calculated using the true distribution q and N samples. Where $p(y_i)$ stands for the predicted probability that a sample i belongs to label y_i. This means that the BCE tells us how high the probability of the data is that they belong to the correct label y. With the BCE and the help of backpropagation the model of the discriminator can be trained. In order to train the generator, we use the label of the real training data for the fake data in the BCE. This enables a gradient to be created that brings the model of the generator closer to the real distribution. The backpropagation is used for the training of the generator as well. By using the right ratio between the two training processes, a generator can be trained iteratively, which at the end is able to represent the distribution of the training data fairly accurately.

Nevertheless, there are many optimizations [MO14, ACB17, ZML16] and challenges that GANs may pose. The aforementioned balancing between the generator and discriminator is often a critical task [Goo17]. Apart from this, there are many other potential difficulties, which are not discussed further here. For more information regarding possible challenges see [Che21], [TTV18], [Bar18].

2.5. Adversarial Attacks

Deep Neural Network models are becoming increasingly popular in a variety of fields. At the same time there is a growing concern about Machine Learning models being attacked in the context of IT security. Adversarial Attacks [SZS+13] refers to attacks that specifically target ML models in order to mislead them. For a ML model M_θ with parameters θ this problem can be defined as [GSS14]:

$$\hat{x} = x + \eta, \quad \text{with} \quad \|\eta\| < \varepsilon \tag{2.13}$$

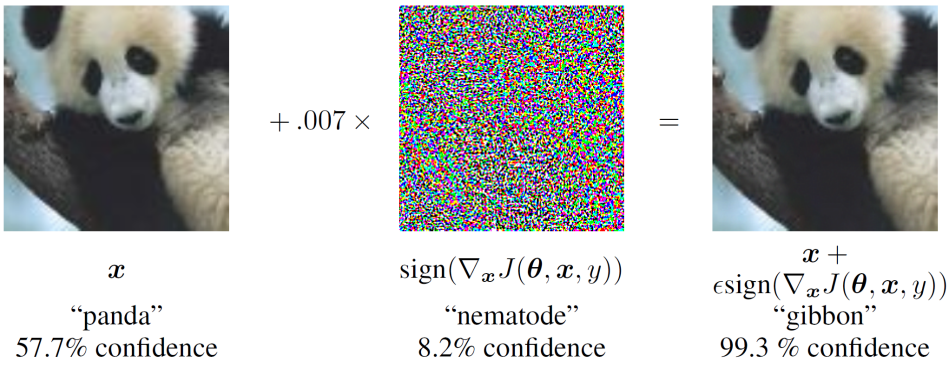


Figure 2.2.: Adversarial Example from [GSS14]: Applying a perturbation η (middle) on an image x (left), causes the resulting adversarial example $\hat{x}$ (right) to no longer be correctly identified by a classifier.

where x is an ordinary input/real data for a model and $\hat{x}$ is an adversarial example, which is a modified version of x with the help of a perturbation η. When passing an input x forward through a model, we obtain an output y representing the classification of x. However, when we modify the very same input x with a perturbation η into $\hat{x}$, we no longer receive y from the model as an output classification. Therefore the following applies: $f(x) = y \neq f(x + \eta)$, where f represents some function which classifies a given input. This perturbation η can be interpreted as an Adversarial Attack designed to fool the model. There are several different categories into which these can be broken down [YHZ+17].

We can distinguish between unintentional and intentional attacks, where the first one usually refers to safety and the second one to security concerns. In the case of unintentional attacks, images may have been taken by dirty sensors [GRBN09]. As a result the captured image can appear like something else without intention. Here it is desirable that the model becomes more resilient against such errors. On the other hand, intentional attacks are often tailored to the target models and try to deceive them on purpose [GSS14]. Therefore it is even more important for the model to defend against these attacks, since high costs or even lives can be at risk, for example an accident can be triggered in the context of autonomous driving.

Another distinction is related to the attacker's knowledge about the model or a lack thereof. If the parameters θ of the model are known to the attacker, the attack is referred to as a white box attack [KS17], otherwise as a black box attack [And17]. The parameters are usually only known to the person who trains the model, thus white box attacks are often used for module/model testing. Whereas black box attacks are more used for testing components of third-party models and therefore are typically applied in the context of integration testing.

On closer inspection of the classification y of the model, it is also possible to distinguish between different types of attacks. Especially in a multi-class classification setting, the category into which the classification y of an input belongs may be very important. In the case where the Adversarial Attack aims to achieve a specific classification category as a result of the perturbation η, we refer to the attack as a targeted one [GSS14]. The goal here is to maximize the probability for an input to be assigned a particular label

by the model. However, if the goal of the attack is mainly to avoid the original label being assigned to the input, we are dealing with a non-targeted attack [SBBR16]. Note that the special case of binary classification implies that the attack is both targeted and non-targeted.

The falsification can also be used to differentiate between attacks [YHZ+17]. When assigning an incorrect classification to a correct/real input, this is referred to as a type II error or false negative. Here the goal is to lower the success rate of the model's classifications regarding real data. We define attacks as type I error or false positive in the case of an attack which aims to classify a incorrect/unreal input from the model to be real data. Therefore we want to trick the model into accepting incorrect data as if they were real.

If we take a look at Adversarial Examples, we can put these into one of two categories, natural [HZB+19] or unnatural [GSS14]. We regard an Adversarial Example as natural if it is no longer distinguishable from the real data, otherwise as unnatural. This is particularly relevant for the security aspect of machine learning models. Unnatural/unrealistic examples may be generated and fine-tuned more easily, however they are identified quickly and without much effort by external parties. This is not always the case with natural/realistic examples, as they are very difficult or even impossible to differentiate from real data. As a result they pose a greater threat as potentially malicious attacks.

We can also distinguish between one-time [HPG+17] and iterative [PGS+20] Adversarial Attacks. With one-time attacks, we perform a single targeted attack and have therefore only the opportunity to optimize this single attack once. On the other hand, with iterative attacks we can repeatedly interact with the model and adapt/improve the attack iteratively to meet our needs.

For the perturbation we have to make a differentiation between constraint and optimized ones, see [YHZ+17]. For constraint perturbations it is only desirable for ε to be sufficiently small, since the perturbation itself is not the primary objective of the Adversarial Attack optimization. However, optimized ones aim to alter the data in such a fashion that it is not possible for an outsider to notice this modification. Hence the perturbations play a very important part in the optimization of the Adversarial Attack.

There are also Adversarial Attacks in the context of Multi-Agent Reinforcement Learning. The most common goal here is to decrease the performance of the agents through modification or suppression of various elements of the system [LHL+17, PDSG17, GY18]. This can contribute towards helping agents becoming more resilient to such attacks. Multi-agent systems offer a wide range of attack vectors. Probably the most popular one is the modification of the observation of agents [TWW+21, XEZJ22], where, for example, the perception becomes partially obscured or is disrupted using noise. Another involves the alteration of the environment itself [RRD+20, HZ19]. This may involve, for example, moving or rotating elements in the environment. As a result, agents have to interact with states they have never seen before. A rather unexplored attack vector are the agents themselves. It is possible that some agents may have been compromised and thus have a potential negative impact on other agents [PGS+20].

2.6. Multi Objective Optimization

Some problems involve not only one objective, but several different ones, which may also be contradictory. Consequently the improvement of one goal is likely to have a negative side effect on another objective. Thus we have to find a trade-off between the different objectives in order to achieve an optimal solution which satisfies all objectives. This challenge is studied in the field of Multi Objective Optimization (MOO) [Gun18]. Here the problem of optimization of different objectives is defined as a minimization of the outcome vector $y_o = f(x_o)$ representing the different goals [Nak05]:

$$\min f(x_o) \equiv (f_1(x_o), f_2(x_o), ..., f_{k_o}(x_o)), \tag{2.14}$$

where $x_{o,d} \in \mathbb{R}^n$ are vectors in the decision space, with $x_o \in X_o$ feasible solutions in the feasible set X_o. These feasible solutions x_o are vectors $x_{o,d}$ which satisfy m_o constraints defined as $g_j(x_{o,d}) \leq 0$ with $g : \mathbb{R}^n \to \mathbb{R}^m$ and $j = 1, ..., m$. From $y_o = f(x_o)$ we obtain the overall evaluation $y_o \in Y_o$ of the feasible solution x_o in regard to all objective evaluations $f_{k_o}(x_o) \in f : \mathbb{R}^n \to \mathbb{R}^p$, where k_o denotes the k'th of p total objectives. Thus the goodness of the solutions can be determined by the dominance. A solution vector $x_a \in X_o$ is said to dominate another solution vector $x_b \in X_o$, if $f_{k_o}(x_a) \leq f_{k_o}(x_b)$ holds for all $k_o \in \{1, 2, ..., p\}$ and additionally $f_{i_o}(x_a) < f_{i_o}(x_b)$ is true for at least one $i_o \in \{1, 2, ..., p\}$. On the other hand, if $f_{i_o}(x_a) > f_{i_o}(x_b)$ holds for any $i_o \in \{1, 2, ..., p\}$, x_a does not dominate x_b. In the case both $f_{i_o}(x_a) > f_{i_o}(x_b)$ and $f_{l_o}(x_a) < f_{l_o}(x_b)$ are true for any $i_o, l_o \in \{1, 2, ..., p\}$ no solution dominates the other, for further information see [Nak05, Gun18, ABC+20]. If we consider the set of feasible solutions which are not dominated by any other solution (Non-dominated solution set), we refer to them as the Pareto-Optimal Set. The Pareto-Optimal Set defines a boundary called Pareto Front, which represents the most optimal trade-off between the different objectives. In Multi Objective Optimization [ABC+20] we are interested in finding the Pareto Front, since the individual $f_{k_o}(x_o)$ are usually not given directly. One difficulty in this regard lies in the shape of the Pareto Front, which is often not convex in the solution space and therefore hard to identify. There are several approaches to address this problem, such as the ε-Constraint Method [YLD71] and Weighted Sum Method [KdW06], although we will not elaborate on them here. In addition to finding the Pareto Front, we want a balance between the different objectives, which is often subjectively preferred by individuals. Because different parties have various preferences in regard to the objectives, finding a clear trade-off which satisfies everyone can be very difficult. Multiple-Criteria Decision Analysis [VH13, ASM+11] is a process designed to help the decision makers balance these conflicting objectives and come up with a trade-off.

3. Related Work

Software testing has become one of the most important parts of software development [AK19]. It involves the testing of software components to detect any possible errors that might be caused either by accidental or intentional attacks. There are many standardized tests, that can be utilized even without specific knowledge about the software execution [JAAA16]. In recent years machine learning algorithms have also become widely used in a variety of applications. However these algorithms have most often only been trained and immediately deployed without being tested for potential failures beforehand [SVJ19]. Because the interpretation of neural network outputs is very difficult or even impossible [HEKK18, GAZ17, ZHML19], we are highly interested in making neural networks as resilient as possible. In this context, Adversarial Attacks serve a crucial part in testing machine learning models against unintentional or intentional threats, because these models are also applied in critical infrastructure, such as transportation [BH22] or energy supply [AFLZ22]. With the help of Adversarial Attacks, it is possible to improve the robustness as well as the resilience of machine learning models using Adversarial Learning and make them less vulnerable to failures and attacks [KS17]. Here the learning process can be described as a zero-sum game 2.3, where two parties pursue opposing goals and train each other using a minmax method. One of the areas where this principle is frequently utilized is in the field of content generation, where a Generative Adversarial Network is typically being used [KLA18, HCC+21, CDS19]. But also in the field of Reinforcement Learning, it is possible to train agents in the same manner and thus improve their performance [PGS+20]. Adversarial Attacks may be used to have a negative impact on various elements of the multi-agent system, in order for the agents to learn to deal with these and therefore to become more resilient.

3.1. Agents as attack vector

In previous work mainly elements in the environment [SCHL21, PDSG17] or the observation [GY18, LHL+17, QYZ+21] were modified or manipulated. However, using the agents themselves as an Adversarial Attack is rather a novelty. In [PGS+20] specifically this aspect has been used as an attack vector in the multi-agent system. The approaches called ARTS and QMixMax were also introduced among other things, which are intended to support agents in becoming more resilient against such failures/attacks. The agents were divided into two groups, the protagonists and antagonists, where the protagonists are normal agents trying to maximize the expected return while the antagonists are the Adversarial Attackers with the focus on minimizing the overall reward. Therefore the goal of the antagonists is exactly the opposite of the one for the protagonists, which makes the process to be represented by a zero-sum game 2.3. Furthermore, the number of attackers has been varied, which has been specified by a so-called antagonist-ratio $R_{ant} = N_{ant}/N$. This ratio represents the proportion of pro- and antagonists in the multi-agent system. The results of the work with regard to this Adversarial Attack revealed

that the performance of the agents, measured by the success rate in the environment, dropped considerably with the increase of the antagonist-ratio when there was no adversarial learning. The reason for this drop, according to the authors of [PGS+20], is caused by the overfitting of agents to the team behavior during the training, resulting in agents blindly trusting in other agents. Hence replacing team members that have been previously trained will result in very substantial performance decreases. However, the introduced approach ARTS has shown that agents can improve their resilience against such arbitrary substitutions through adversarial learning. As a result it is possible to counteract any potential outages or other attacks to a certain degree.

Similarly to [PGS+20], in our work we will also be focusing on the team agents as the attack vector. Nevertheless the primary goal of our work is not to increase the resilience of agents against Adversarial Attacks, but rather to provide a test suite against malicious attacks. The focus here will be on antagonists that not only try to minimize the overall reward, but shall also remain undetected by any third party for a long period of time. For this purpose we will take a closer look at a number of different Reinforcement Learning algorithms as targets, while using a black box attack to represent a highly realistic scenario. In contrast to [PGS+20], the environment will be a Dec-POMDP, in which we will face highly cooperative scenarios, and substitute cooperative agents by Adversarial Attackers.

3.2. Hidden Adversarial Attacks

Hiding Adversarial Attacks has already been the main focus in some previous works [ZDS17, DMW+20, BSGK22]. Attacks were hidden by placing them in areas that were inconspicuous to observers, but very noticeable within them [SVS17]. However, in [GWH19], under the consideration of human observers, images were created using perturbations, which are harder to distinguish from the original compared to common adversarial examples. For this, the authors did not concentrate the attack at one spot, but spread it across the canvas. In order to hide the attack, a strength map ξ, indicating the intensity of the perturbation on the image, was used. For non-hidden perturbations, the strength map might be uniform, however, this does not reflect the perception of a human being, since humans tend to focus on specific regions. Therefore, under consideration of human perception biases [Pro07], the strength map in [GWH19] has been based on the Shannon entropy [Sha48]. For each pixel i, j the local Shannon entropy $SE_{i,j} = \sum_{p \in B_{i,j}} p \log p$ over the local neighborhood density B has been calculated. These pixel entropies were subsequently used as the strength map ξ and combined with the help of the Basic Iterative Method (BIM) [KGB16] to provide a new method called Entropy-based Iterative Method (EbIM) [GWH19]. Through this EbIM Adversarial Attacks were created which were hidden more effectively.

Similar to [GWH19] we also intend to hide our Adversarial Attacks in this work with regard to external observers, yet we do not limit to or tailor them to human ones. In addition we aim to localize our attacks to some extent, which is done using a reward threshold instead of a computationally intensive mapping. Finally our goal is not to attack the classification of images, but rather to disrupt the complex behavior of agents in a multi-agent system. Hence the main goal of this work is to hide the antagonists from external observers while at the same time minimizing the overall performance of agents.

4. Experimental Setup

Reinforcement Learning algorithms are typically evaluated using environments with agents which combined represent a MDP [Bel57]. These environments are approximations or highly simplified representations of the real world [SMRL59, LZL+17, LMK+17]. In the context of agent behavior, we refer to environments as games, which originates from game theory [CEW11]. Games provide a great platform for studying various aspects of agent behavior [CEW11]. Over the past several years, single agent environments [Net05] have developed into more interesting settings [MKS+13] beyond toy examples like grid worlds [CPLK20]. These single agent environments are becoming more and more important for the real world as they cover ever increasing realistic aspects. Consequently there are many agents that have been trained in a simulated setting that are being deployed in the real world, such as in autonomous driving [VAAMS22]. However many problems involve multiple agents and can therefore only be represented in a multi-agent system. One of the most important aspect here is the cooperation or non-cooperation between agents [HBD21], where agents have to work as a team in order to achieve success in the domain. There are many works that have studied the social behavior of agents in such games, since many challenges of this kind emerge in multi-agent settings [BEH+20]. Especially in these challenges different algorithms can compete against each other and for this reason games also provide a large testbed and benchmark tool in the field of Reinforcement Learning. One game in particular that has received a lot of attention in this regard in recent years is the StarCraft Multi-Agent Challenge (SMAC) [SRdW+19]. It offers a number of different cooperative scenarios, which we will take a closer look at later on.

4.1. StarCraft Multi-Agent Challenge

The environment we use for our approach is the Starcraft Multi-Agent Challenge (SMAC) [SRdW+19]. It is based on the popular real-time strategy (RTS) game Starcraft II, but is distinct from the game itself. There are two general parts of Starcraft II that need to be learned: the Macro Management and the Micro Management. In Macro Management, the objective is to control the game in its overall sense [WZH17]. Here, for instance, the economy, the building of units, but also the scientific research have a significant impact on the outcome of the game. Micro Management, on the other hand, focuses for the most part on the units in the game [SZZ18]. This involves the positioning and the action selection of the units on the battlefield. The SMAC centers around the micro management aspect of Starcraft II and provides scenarios with a decentralized manner with regard to the agents. In contrast to the complete RTS game as it is realized in [VBC+19], the SMAC neglects the macro management. The SMAC is a Dec-POMDP (see section 2.1) which makes it a very realistic game in terms of the agent system. In recent works, there has been a lot of focus in the Reinforcement Learning community to use this environment as the benchmark for algorithms for cooperative agents in multi-agent systems. This is in part because the micro management in this environment has a very high skill ceiling,

allowing various algorithms to measure themselves against each other. In the environment two armies face one another and have to defeat their opponents [SRdW+19]. The agents control one side in the battle and have to learn how to destroy the hostile units, which are controlled by a hand-crafted AI. For our purposes, we set the difficulty of the enemy units to level 7, which is considered to be very difficult. In order to defeat the enemies, each agent in the multi-agent system has to learn independently to cooperate with the other agents, while they are only able to perceive their immediate, partially observable surroundings. Through an attack action, it is possible for the agents to cause harm to their enemies, initially to their shields, if they have any, and subsequently to their health. When the health of any unit reaches 0, that unit is considered to be defeated and is taken out of the game. Note that shields will regenerate if the unit has not been attacked for a period of time and that all active special abilities in the SMAC have been disabled. With the proper micro management, the damage to enemies can be maximized while at the same time minimizing the harm to agents. Once a team is entirely defeated, the episode of the Dec-POMDP ends. Should the time limit be exceeded beforehand, then the learning agents will lose automatically. The goal of the environment for the agents is to maximize their win rate. A more detailed description [SRdW+19] of the environment's observations, states, action spaces, rewards and the individual scenarios as well as their challenges is provided in the following.

4.1.1. Action Space

In the SMAC the agents are given a discrete set of actions from which they can choose [SRdW+19]. However, due to the state of the environment, there may be some actions which cannot be taken as these are not possible. The particular actions which cannot be chosen are communicated to the agents through the observation. The discrete set of actions consists of Movement, Attack, Stop and No-Op actions. An agent can take the Movement actions to move in one of the four directions: north, south, west and east. Depending on whether a direction is blocked, the corresponding action may not be available to the agent. When selecting an Attack action, the agent must choose one of the enemies to be targeted. If the target of the attack is not within attack range, this action cannot be executed. Here the individual units determine how large their attack range is. Ranged units have a significantly higher range compared to a melee unit. If a unit is only capable of Healing, the Attack action becomes the action used for Healing. When an agent decides to take the Stop action, that agent becomes inactive for one timestep. The No-Op action can only be performed by agents who have already been defeated. Moreover, they cannot choose any other action except this one. The No-Op action does not have any effect on the environment.

4.1.2. States and Observations

The SMAC [SRdW+19] does not provide the agents with insight into the states of the environment during the execution phase. However, they are made available during the centralized training phase of the agents. The environment states are composed of the following components: coordinates, health and shields of all units as well as the energy and cooldowns of the agents. Here the energy indicates the remaining healing capacity of the unit Medivac and the cooldown the time until the next attack action of an agent

is available. Additionally, it is possible to include for each time step the action of the last step in the state as well. Besides the states, the environment also has a tailored observation for each agent at each time step. However, this observation only contains information within the immediate sight range of an agent (in this domain the sight range is set to 9). The sight range is specified individually for each agent using a circle with the center at the position of the agent. Note that other units which are located outside the sight range are not included in the observation and consequently cannot be seen by the agent directly through the observation whether these units are still alive or their exact position. Therefore the environment is only partially observable by the agents 2.1. The observation at one time step for an agent contains the available actions, the terrain height and if in sight also the relative coordinates, health, shields and the actions at the previous time step of the allies, enemies and the agent itself.

4.1.3. Rewards

In this environment, agents are rewarded if they inflict damage on enemies [SRdW+19]. Depending on the scenario and the number of agents, the reward amount may vary for one successful attack. Additionally the agents are given a greater reward if they manage to defeat all of the enemies and thereby achieve a victory. In the SMAC the agents will collectively receive all of the rewards. For instance, if only a single agent deals damage to an enemy, then all of the agents are given a reward. The SMAC also provides a sparse reward variation, where agents are only rewarded at the end of each episode with either +1 reward, for a victory, or -1 reward, for a defeat. While these sparse rewards would pose an interesting challenge for an extension of our approach, we will not use them in this work. Besides these rewards, the SMAC also offers the possibility to penalize the agents in case they suffer damage. However we will not use these penalties because the agents should autonomously learn to survive in order to achieve a victory.

4.1.4. Scenarios

There are a number of different scenarios provided by the SMAC [SRdW+19]. In these the agents have to learn a specific technique for each scenario in order to win. The scenarios themselves can be characterized by certain properties. There are homogeneous scenarios in which all units of the agent team have the same unit type, otherwise it is referred to as a heterogeneous scenario. Furthermore some scenarios are symmetric, whereas this property is related to the composition of the enemy team. If the two parties do not have the same units, we refer to this as an asymmetric scenario. Those are considered to be more difficult since the agents in these scenarios are usually outnumbered. In the following subsection, a more detailed description is given of the scenarios we use for our work.

3 Stalkers vs 3 Zealots

The 3 Stalker vs 3 Zealots (3s vs 3z) scenario is a homogeneous and asymmetric scenario. In it the agents control three units of the type Stalker, who can deal damage to enemies from a distance, making them ranged units. They fight against three of the handcrafted AI controlled Zealots, which can only attack in close combat (melee units). Both of the unit types belong to the Protoss race [XI15] and therefore have a shield in addition to the

Figure 4.1.: The 3 Stalker vs 3 Zealots environment: Agents control the ranged Stalkers (left) and have to defeat the melee Zealots units (right).

regular health. While the Stalkers are equipped with 80 health and 80 shield, the Zealots have 100 health and 50 shield. Although the Stalkers endure more damage in total, the Zealots have a significant advantage over the Stalkers in another aspect. The attack speed of the melee unit is much faster compared to the Stalkers, therefore the latter would lose if they never moved. In order to achieve a victory in this scenario, the agents have to learn a micro-trick called kiting [UO12]. By kiting, agents keep their opponents at a safe distance while they attack the enemy. This requires the agents to learn to use directed movement actions that lead away from the enemies in a regular interval, alongside the pure attack actions. Because of this particular challenge, this scenario is rated with the difficulty easy.

5 Marines vs 6 Marines

Figure 4.2.: The 5 Marines vs 6 Marines environment: While the agents control 5 Marines units (left) which are outnumbered, they have to fight their enemies who consist of 6 units (right).

The next scenario we are interested in is the 5 Marines vs 6 Marines (5m vs 6m) setting, which is homogeneous and asymmetric. In this scenario the agents are in control of the Marines, who are outnumbered, and have to defeat the handcrafted AI units. The Marine unit is a ranged unit, without any shields, and belongs to the Terran race. Since

the agents are outnumbered with 5 to 6, they have to cooperatively coordinate themselves to achieve a victory in this scenario. This is where the micro-trick focus fire [SRdW+19] plays a crucial part in defeating the enemy. Here the agents have to focus their attacks on a single enemy unit in order to remove the target as fast as possible from the game and thus overcome the disadvantage in the number of units. Part of what makes this micro-trick challenging is the coordination of agents on a collective attack target. This becomes difficult as the agents are not able to explicitly exchange information through direct communication. Hence each agent has to individually learn to make a decision regarding the attack target, while still being consistent at the same time with the other agents. Simultaneously the agents have to be careful to avoid the enemies using the same trick. To prevent this from happening, it is necessary to distribute the damage of the enemies among all agents, with the purpose to keep them alive for as long as possible. This requires the agents to be aware of their own health and to recognize when they are under attack. Depending on the situation, a corresponding agent has to retreat temporarily to transfer the incoming damage to other allies. The execution and prevention of this micro-trick pose a great challenge. The scenario itself is therefore rated with the difficulty hard.

1 Medivac, 2 Marauders & 7 Marines

Figure 4.3.: The 1 Medivac, 2 Marauders & 7 Marines environment: Two identical teams each with one healing unit Medivac have to destroy the other. During the battle the agents control one party (left) and the handcrafted AI the other (right).

For the last scenario which we want to focus on in this work is the 1 Medivac, 2 Marauders & 7 Marines (MMM). This scenario is heterogeneous and symmetric [SRdW+19]. Here the agents have to control one of two identical teams and combat against the very same units which are controlled by the handcrafted AI. During this battle Marauders, Marines as well as the Medivac are not equipped with any shields, instead they are only provided with health. However the Medivac has the ability to fly and therefore can not be attacked by the Marauders. Similar to the previous scenario 5 Marines vs 6 Marines, the agents have to learn the micro-trick focus fire [SRdW+19] in order to be successful in this environment. Nevertheless, this scenario is overall easier, as the agents here are not outnumbered. Despite its simplicity, there is one interesting aspect which makes this

scenario quite interesting for our purposes: the healing unit Medivac. As this unit is not able to deal any damage to the enemies and thus can not acquire any reward directly. Instead, the Medivac can only contribute to the success of the agents in this setting indirectly through healing the other team members. However overall this scenario is rated with the difficulty easy.

5. Infiltrating Stealth Agent Attack Controller (ISAAC)

There are many work on Adversarial Attacks in multi-agent systems, however these mainly focus their attack on the observation [ZCBH21, LHL+17, GY18, XEZJ22] or elements within the environment [PDSG17, HZ19, RRD+20]. In our work we use the agents themselves as an attack vector. Until now the agents in multi-agent systems have been assumed to be without any flaws, like in [SKK+19, RSdW+18, SRdW+19]. However, in real world applications they may fail due to flaws like broken parts. Another reason for agent misbehavior or failure might involve an Adversarial Attack. Such attacks can compromise agents to be no longer functional or even sabotage the performance [PGS+20]. In this case, invaders are usually easily distinguishable from the actual functional agent [LLM+22]. Therefore an external attacker might attempt to hide such an attack from observers [GWH19]. In this work we will take a closer look at a possible Adversarial Attack which not only decreases the overall performance, but also tries to hide the attack. For this purpose we will only consider fully trained agents in a multi-agent system as targets. In order for this scenario to be as realistic as possible, we design our Adversarial Attack to be natural, iterative, optimized, intentional, targeted, false positive as well as to be a black box attack 2.5. Hence we intentionally compromise agents in the environment to learn to iteratively degraded the overall performance, while we optimize the perturbation to be natural by making the compromised agents appear to be the targeted common ones. As a result any external observer is expected to mistake these false compromised agents for positives. At the same time, we intend to treat the scenario as a box black attack, in which we do not have any knowledge about the model weight parameters of the agents. In the following we refer to the compromised agents as ISAAC Invaders and the common agents as Protagonists. ISAAC consists of an imitating and an antagonistic part. Here we use the antagonistic part to decrease the performance of the protagonists. Agents who pursue only this objective are referred to as Naive Invaders. For this purpose we utilize the same algorithms which are used by the protagonists for the attackers 2.2. On the other hand we hide the attacking agents by imitating the protagonists in multiple aspects. In this context the history experiences $\boldsymbol{e}_t^{\boldsymbol{\tau}}$ as well as the states of the environment s_t play a crucial part. Throughout this work we will only focus on the training of the attackers. We will describe these components in greater detail and explain how they interact with each other in the following sections.

5.1. Antagonistic Behavior

One of our goals for the ISAAC Invader is to decrease the performance of all agents in an arbitrary environment. To accomplish this, we consider, as in [PGS+20], the multi-agent system as a zero-sum game 2.3, in which the decrease of one party simultaneously translates into the gain of the other. We split the supposedly cooperating agents $\mathcal{S}_{tot}^{a}$

with $|\mathcal{S}^a_{tot}| = n$ into the two groups for our zero-sum game. Here the game is composed of the set of protagonists $\mathcal{S}^a_{pro}$ with $|\mathcal{S}^a_{pro}| > 0$ and the set of the ISAAC Invaders $\mathcal{S}^a_{atk}$ with $|\mathcal{S}^a_{atk}| > 0$, such that the following applies: $\mathcal{S}^a_{pro} \subset \mathcal{S}^a_{tot}, \mathcal{S}^a_{atk} \subset \mathcal{S}^a_{tot}$ and $\mathcal{S}^a_{pro} \cup \mathcal{S}^a_{atk} = \mathcal{S}^a_{tot}$ with $\mathcal{S}^a_{pro} \cap \mathcal{S}^a_{atk} = \emptyset$, see [PGS+20]. This enables us to view both sets as opposing, since they optimize two conflicting objectives. While the protagonists aim to maximize the objective of the environment, the antagonists strive to minimize it. Therefore we can define this counter-play as a zero-sum game 2.3. While many environments are designed to be a zero-sum game, we artificially create such a sub-game inside any environment as a result of deploying the ISAAC Invaders. Since the protagonists use the shared joint reward function $r(s_t, \boldsymbol{u}_t)$ of the environment, we can modify the output reward for the antagonists due to nature of zero-sum games, that a loss corresponds to a gain of the other party 2.3. Hence if an agent receives a reward R_t from the shared joint reward function $r(s_t, \boldsymbol{u}_t)$, we can interpret this as a penalty for the attacker. Therefore we define the reward of the protagonists as $R_{pro,t} = r(s_t, \boldsymbol{u}_t)$ and the corresponding reward for the attackers as $R_{atk,t} = -r(s_t, \boldsymbol{u}_t) = -R_{pro,t}$. Consequently we can keep the shared joint reward function $r(s_t, \boldsymbol{u}_t)$ for both protagonists and attackers.

One benefit we gain from splitting the set of agents into two teams, is that we can still utilize the Centralized Training with Decentralized Execution (CTDE) paradigm 2.2. Therefore we can train arbitrarily large teams in a multi-agent system and fulfill the IGM requirement 2.2 within the parties. However, the joint global history experiences $\boldsymbol{e}^{\boldsymbol{\tau}}_t$ we gain from the environment contains the actions of the protagonists $\boldsymbol{u}_{t,pro}$ and the antagonists $\boldsymbol{u}_{t,atk}$. This might cause problems in the learning process, as we separate the joint policies of the protagonists $\boldsymbol{\pi}_{pro}$ and the attackers $\boldsymbol{\pi}_{atk}$ into two parts and need to update them independently in regards to the teams. Hence we need to take this combined joint global history experiences $\boldsymbol{e}^{\boldsymbol{\tau}}_t$ into account when we calculate the joint action-value function $Q^{\boldsymbol{\pi}}(s_t, \boldsymbol{u}_t)$ and the joint policies $\boldsymbol{\pi}$ of both teams. In order to only update the policies $\boldsymbol{\pi}$ with the experiences of the corresponding teams, we make use of the IGM and set all individual action-value functions $Q^{\pi}_i(\tau_i, u_i)$ to zero, if the agent i does not belong to the team of which we calculate the joint action-value function ($Q^{\boldsymbol{\pi}}_{pro,tot}(\boldsymbol{\tau}_{pro}, \boldsymbol{u}_{pro})$, $Q^{\boldsymbol{\pi}}_{atk,tot}(\boldsymbol{\tau}_{atk}, \boldsymbol{u}_{atk})$): $Q^{\pi}_{pro,i}(\tau_i, u_i) = 0$, if $i \notin S_{pro}$, $Q^{\pi}_{atk,i}(\tau_i, u_i) = 0$, if $i \notin S_{atk}$. Thus we avoid mixing the individual action-value functions of the protagonists $Q^{\pi}_{pro}(\tau_{pro}, u_{pro})$ and the attackers $Q^{\pi}_{atk}(\tau_{atk}, u_{atk})$. By using the separate joint action-value functions of both teams $Q^{\boldsymbol{\pi}}_{pro,tot}(\boldsymbol{\tau}_{pro}, \boldsymbol{u}_{pro})$ and $Q^{\boldsymbol{\pi}}_{atk,tot}(\boldsymbol{\tau}_{atk}, \boldsymbol{u}_{atk})$, we can minimize the loss defined in 2.1 for the protagonists and the attackers. Another benefit which we gain from this separation of protagonists and attackers into two sets is that we can use different action-value functions for both teams 2.2. This allows us to use different algorithms to decrease the performance of the protagonists, which might be more effective compared to using the same action value function.

5.2. Protagonist Imitation

The main goal of our work is to hide the Adversarial Attacker from external observers, while also decreasing the overall performance. In order to properly hide an attacker, we have to consider two aspects in particular: The first thing an agent has to accomplish to stay unnoticed is to study and mimic the behavior of the protagonist 5.2.1. Another aspect for successfully blending an attacking agent into the system is to replicate specific parts of

the environment states s 5.2.2. While imitating the protagonist alone would be sufficient if the critical observation is made from the point of view of the agent itself, from an external viewpoint we can immediately recognize the copycat [LLM+22]. Consequently, it is necessary for the attacker to have a strong guidance with regard to some parts of the environment in order to avoid being exposed. Since we are interested in substituting a particular protagonist, we also require information related to the protagonist's behavior and the state progression throughout the episode, compared to the antagonist part. In the antagonistic component 5.1, we do not need any additional interactions with the substituted protagonist, since the antagonist learns to minimize the reward of all agents directly by engaging with the environment. During the learning process to imitate the protagonist, it is necessary to repeatedly interact with the substituted agent and also record episodes with no attackers in order to obtain more information with regard to the states s visited. In doing so we use the substituted protagonist as a real data generator 2.4. By applying the observations z we obtain from our environment during the learning process as inputs for the protagonist, we receive the actions the protagonist $\boldsymbol{u}_{pro}$ would have taken. Similarly to the protagonist's choice of actions for given observations, we are able to learn about the states s that the protagonist visits during the episodes without any attackers. Consequently by utilizing specific features of the states, such as the positions, we are able to give our attacker a reward, given that the attacker is able to reproduce those parts of the states in the environment. In the following sections we are going to take a more in-depth look at both of these components of the ISAAC Invader, as well as the difficulties which they may pose.

5.2.1. Imitate Behavior

In order to hide our ISAAC Invader with respect to the action selection it is necessary to study the behavior of the protagonist and on the basis of this learn how he interacts with the environment. To adapt our actions $\boldsymbol{u}_{atk}$ to those of the substitute protagonist $\boldsymbol{u}_{pro}$, while simultaneously executing an attack against all other agents, it is necessary to mimic individual intentional actions. For the learning process of our ISAAC Invader regarding the actions to be executed, we make use of a GAN structure 2.4, which includes a classifier to discriminate between the actions of the protagonist $\boldsymbol{u}_{pro}$ and those of the attacker $\boldsymbol{u}_{atk}$. When we provide an observation z to the substituted protagonist and the attacker, both of them return probabilities on which action the respective agent would select under this observation. Here, the chosen action may be discrete, with a singular value, or continuous, composed by a grouping of values representing all of the possible actions [ZWZ22]. Furthermore in this case we do not require any of the parameters θ of the agent models, allowing us to treat the attack as a black box attack 2.5. For teaching the attacker the actions of the protagonist we employ a GAN 2.4. An ordinary GAN [GPAM+14] consists of a generator and a discriminator, whereas the generator produces fake data while the discriminator is supposed to distinguish between these and real data. The goal of any GAN is to approximate the underlying distribution of the real data by utilizing the generator. At the end of the training process, the generator is expected to produce data which is closely resembling the real data. In our work, besides other information, we use the action selections made by the agents $\boldsymbol{u} = \boldsymbol{u}_{pro} \cup \boldsymbol{u}_{atk}$ as input data which we pass to the discriminator. In this regard, we consider the individual action choices of the protagonists $\boldsymbol{u}_{pro}$ and the attackers $\boldsymbol{u}_{atk}$ as separate data points that we sample

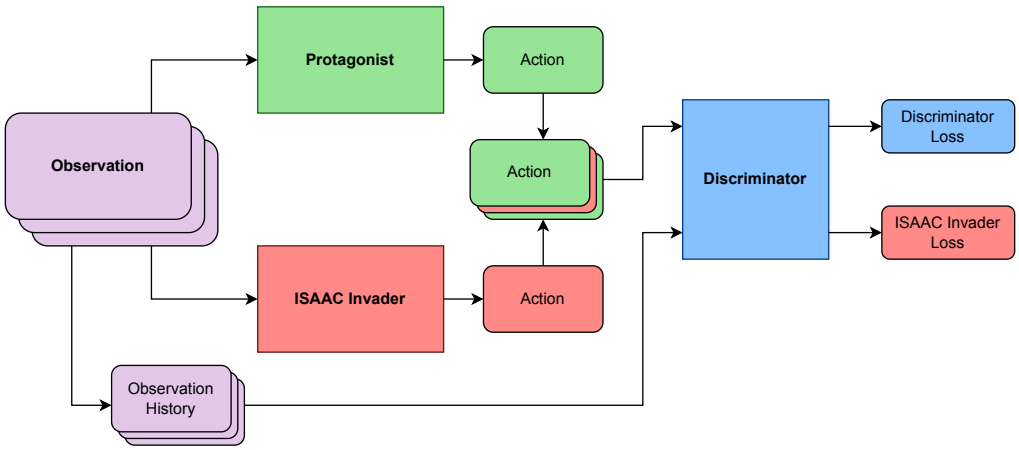


Figure 5.1.: Behavior Imitation Information Flow: The observations are passed to the substituted protagonist and the ISAAC Invader who output the respective actions. Afterwards the labeled actions are fed to the discriminator along with the observation histories. At the end we compute the respective losses for the ISAAC Invader and the Discriminator.

out of distributions, also known as the policies $\boldsymbol{\pi}_{pro}$ and $\boldsymbol{\pi}_{atk}$ 2.1. These distributions represent the behavior of the respective agents, which are conditioned, but not limited, to the current observation z of the environment. Therefore we treat the actions which were chosen by the substituted protagonist $\boldsymbol{u}_{pro}$ as real data (see 2.4). On the contrary, the hidden attacker serves as a kind of generator, who produces fake data by his selection of actions $\boldsymbol{u}_{atk}$. During the training process the discriminator is trained to distinguish between the actions $\boldsymbol{u}$, the outputs of the agent models when passing observations z, of the protagonist and those of the ISAAC Invader. For that purpose the discriminator also requires the corresponding observations z of the environment alongside the actions $\boldsymbol{u}$. Instead of taking the basic observations z of the environment into account, we choose the observation histories $\boldsymbol{\tau}$ in order to take the sequence of observations over time into consideration as well. As a result, it becomes easier for the discriminator to differentiate between the protagonists and the attackers [YJvdS19], thus contributing further to the learning process of impersonation. Since the observation history $\boldsymbol{\tau}$ takes all of the previous observations z from a certain point in time into account, this may contribute additional information for the classification of the actions. Once the discriminator has made a decision regarding where the actions originated from based on the actions $\boldsymbol{u}$ and the observation histories $\boldsymbol{\tau}$, we can compute the difference between these distributions, the behaviors of the agents, with the help of a value function $B(G, D)$ 2.4. Note that the discriminator does not make a binary decision while classifying the actions, but rather returns probabilities. For calculating the difference between both distributions, which represent the agent policies $\boldsymbol{\pi}$, in our work we make use of the Binary Cross Entropy (BCE), see 2.12:

$$H_p(q) = -\frac{1}{T}\sum_{t=1}^{T}\left[y_{v_t}\cdot\log(p(y_{v_t})) + (1-y_{v_t})\cdot\log(1-p(y_{v_t}))\right], \quad \text{with} \quad v_t = (u_t, \tau_t) \tag{5.1}$$

where $v_t = (u_t, \tau_t)$ denotes the input of the discriminator consisting of the action u_t and the observation history τ_t at time step t.

By minimizing this value function $B(G, D) = H_p(q)$, we can subsequently use the output loss in order to train our ISAAC Invader model. For this purpose we only use the fake data in the BCE and flip the labels, such that we can approximate our ISAAC Invader actions $\boldsymbol{u}_{atk}$ to those of the protagonists with the help of the loss 5.1. Maximizing the value function $B(G, D)$, on the other hand, results in the loss for our discriminator, allowing it to improve its ability to distinguish between the actions $\boldsymbol{u}$. With the help of backpropagation [And86] we can update the parameters θ of the networks and approximate the behavior over the course of the training to that of the protagonist. However, we are not interested in converging to the entire behavior of the protagonist, which is why we force this gradient to be zero for selected time steps t. Especially critical actions, where the protagonists gain a high reward, are to be avoided from the point of view of the ISAAC Invader. For this reason, we filter loss depending on the rewards.

Behavior Shaping

Since some actions may be essential for the success of the protagonists, the ISAAC Invader is not supposed to imitate any of them. For this reason, we apply a mask which enables us to filter the BCE loss where the reward $R_{pro,t}$ obtained by the protagonist would be too high. Through the shared joint reward function $r(s_t, \boldsymbol{u}_t)$ we receive the reward for each time step t of the environment and can repurpose them for our purposes. With the help of the rewards R_t along side a hand-defined threshold T_{bs}, we can filter our loss by applying the following mask:

$$H_{bs,p,t}(q, R_t) = \begin{cases} H_{p,t}(q) & \text{if } R_t \leq T_{bs} \\ 0 & \text{otherwise,} \end{cases} \tag{5.2}$$

where $H_{bs,p,t}$ denotes the filtered loss of the binary cross entropy $H_{p,t}(q)$ from 5.1 with the reward R_t at time step t and a fixed reward threshold T_{bs}. As a result, the actions of the ISAAC Invader $\boldsymbol{u}_{t,atk}$ are not encouraged, which contribute too heavily to the success of the protagonists. During the interaction of the attacker with the environment, the observations z may differ greatly from the ones of the previously trained protagonist. This allows the ISAAC Invader to exploit out-of-distribution states s and observations z of the protagonist during the imitation training thus maximizing the antagonistic part [SB98]. In order to prevent this from happening, it is necessary to guide the ISAAC Invader in such a way that he only interacts with states s that are close to the ones which have been explored by the protagonist. In the following section we elaborate further on how we address this particular issue.

5.2.2. Mimic State Features

For the purpose of hiding our ISAAC Invader in addition to merely copying the behavior of the protagonist during the interaction with the environment, we also need to replicate certain features of the states that have been sufficiently explored by the substituted protagonist. A simple mimicking with no guidance would be sufficient provided that the attacker does not pursue any kind of antagonistic objective [HE16]. However, such an antagonistic goal leads to a Multi Objective Optimization 2.6 in which we want to

minimize the rewards on the one hand and at the same time avoid being detected on the other hand. With no guidance the attacker enters states s which are under-explored by the protagonist, thus exploiting that the protagonist has not learned the optimal actions in order to maximize his action-value function Q^{π}.

In the training process of any agent in an environment, a trade-off between exploration and exploitation has to be found [VRQI10]. During the exploration phase, the agent initially searches for different states s before exploiting the ones which maximize his action-value function Q^{π} during the exploitation phase. Due to the fact that the entire state space $\mathcal{S}$ of the environment usually can not be fully explored by a Reinforcement Learning agent, there are a lot of states s and thus a lot of observations z which the agent has not or barely encountered. As a consequence the agent lacks the necessary experience to maximize his reward in the out-of-distribution states and therefore the policy $\boldsymbol{\pi}$ performs sub-optimal in those states s because it is based on the unexplored observations z. Thus the ISAAC Invader without any guidance can exploit the sub-optimal policy $\boldsymbol{\pi}$ of the protagonist by entering these states himself. Since the sub-optimal policy $\boldsymbol{\pi}$ of the protagonist does not necessarily maximize the joint action-value function $Q^{\boldsymbol{\pi}}(s_t, \boldsymbol{u}_t)$ of the protagonist, the ISAAC Invader barely needs to make a trade-off between the imitation and antagonistic actions. This however leads to a situation in which the ISAAC Invader can be easily spotted, which does not reflect the goal of our thesis. To counteract this we apply a reward shaping in order to keep the attacker closely to the states which the protagonist has sufficiently explored allowing us to better hide the attacker.

For this purpose we use another classifier to distinguish between specific components of the states encountered by the protagonist and the attacker. It is necessary to only consider parts of the states s'_t, as a full replication would cause the ISAAC Invader to no longer be able to minimize the global reward R_{pro}. Hence we need to choose components of the states s'_t (feature selection) which do not conflict with the antagonistic objective. In order to measure the difference between these features, we make use of the Binary Cross Entropy (BCE), like in 5.1. In this process the state components of the protagonists are again treated as real $y_{s'_t} = 1$ and those of the ISAAC Invader as fake $y_{s'_t} = 0$ data. Note that the states s for the protagonists were obtained during episodes which were performed with no attackers. When we pass the features along with the time step t through the classifier, we obtain the predicted labels $p(y_{s'_t})$, which indicate the probability, according to the classifier, that the state components s'_t belong either to the protagonist or the attacker. Afterwards we apply the BCE loss 2.12 in order to train our classifier and incorporate the predicted labels $p(y_{s'_t})$ to shape the rewards for the ISAAC Invader $r_t(p(y_{s'_t}))$:

$$r_t(p(y_{s'_t})) = \begin{cases} r_t + F & \text{for } p(y_{s'_t}) \geq 0.5 \\ r_t - F & \text{for } p(y_{s'_t}) < 0.5, \end{cases} \tag{5.3}$$

where F denotes a reward for deceiving the classifier.

If the features of the state s'_t at a given time step t are assigned by the classifier to be real during the exploration phase of the ISAAC Invader, we increase the reward R_{atk} at this time step for the attacker by F. On the other hand, in case the classifier correctly identified them as fake, the attacker receives a penalty of F at that time step t. Thereby we create an incentive for the ISAAC Invader to act sufficiently close to the states s which the protagonist has already explored, and thus to hide from external observers.

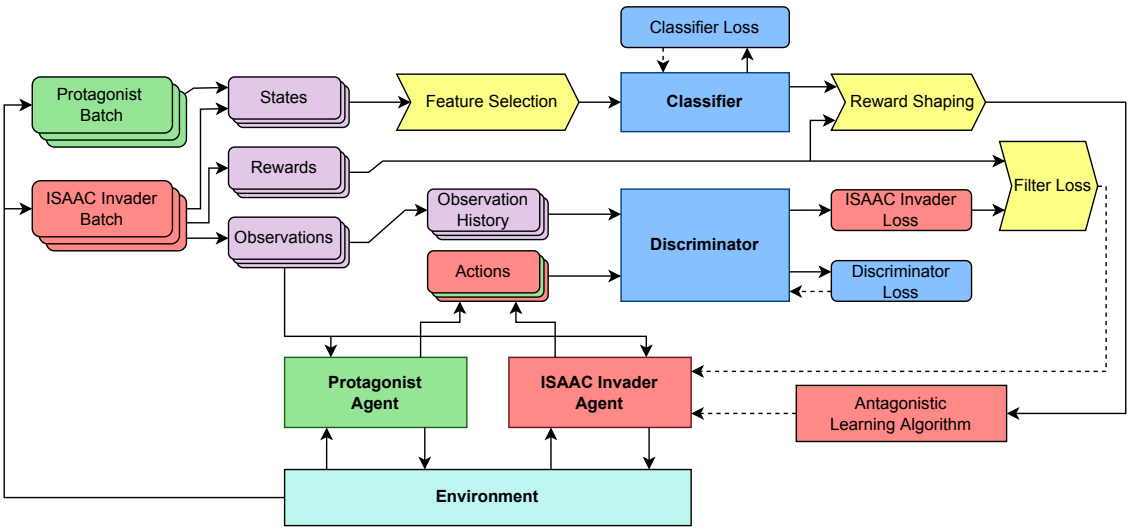


Figure 5.2.: Complete ISAAC structure and information flow: Through the application of the Behavior Imitation Discriminator and the Mimic Feature State Classifier, we compute the corresponding loss for the hiding part. When combined with the antagonistic training, we get the respective update for our ISAAC Invader.

5.3. ISAAC Invader

The complete ISAAC Invader is composed of the antagonistic 5.1 and the imitation components 5.2.1, 5.2.2. While the antagonistic part tries to minimize the overall reward of all the protagonists R_{pro}, the imitation elements focus on hiding the attacker. By copying the behavior of the substituted protagonist $\boldsymbol{\pi}_{pro}$ as well as some elements of the state s', the ISAAC Invader is supposed to sabotage the setting without being caught. Since the ISAAC Invader deals with a Multi Objective Optimization Problem 2.6, we have to find a trade-off between the hiding and the sabotaging. For this purpose we use two classifiers as well as the minimization of the overall reward. During the training both the classifiers and the antagonistic part have to be updated in a balanced ratio, as otherwise either one of the two ISAAC components becomes the dominant one and consequently we obtain an obvious antagonist or a protagonist out of the ISAAC Invader. To avoid such a scenario (also see challenges in 2.4), we utilize a number of different schedules, which should help us to balance out the individual components by training them in specified intervals. While we keep the antagonistic part 5.1 in every training iteration, the Behavior Imitation portion 5.2.1 is only used in a specified manner. We define the ratio of the frequency with which we train the antagonistic part compared to the behavior imitation one as the behavior imitation ratio *bir*. In accordance with this we introduce a Behavior Imitation Training Scheduling Sd_{bir} to ensure that the *bir* is satisfied during the training. The *bir* should prevent convergence problems [SGZ+16, Mes18] or mode collapses [TTV18] from occurring, for which GANs 2.4 are notorious. Furthermore we use a Mimic Feature Training Scheduling Sd_f to avoid under- or overfitting of the state feature classifier. This schedule determines the frequency at which the classifier is trained compared to the agent. How often protagonist episodes are run in comparison to ISAAC

Invader ones are determined by the ISAAC Invader Episode Scheduling Sd_{epi}. Through these three schedules we try to provide a stable training for the ISAAC Invader.

By combining all the concepts introduced in this chapter, we can create the ISAAC Invader who is capable of compromising and sabotaging a multi-agent system unnoticed.

Algorithm 1 Infiltrating Stealth Agent Attack Controller (ISAAC)

Initialize Protagonist and ISAAC Invader replay memories D_{pro}, D_{atk}
Load trained protagonist parameters θ into θ_{pro} for π_{pro}
Initialize θ_{atk} for π_{atk} with random parameters θ_{rand}
Initialize θ_{dis} and θ_{class} with random parameters θ_{rand}
Initialize Schedules Sd_{epi}, Sd_{bir}, Sd_f
Create attack set $\mathcal{S}^a_{comb} = \mathcal{S}^a_{pro} \cup \mathcal{S}^a_{atk}$

for epi = 1 to M **do**
  $\mathcal{S}^a, D \leftarrow$ select $\mathcal{S}^a_{comb}, D_{atk}$ or $\mathcal{S}^a_{pro}, D_{pro}$ according to $Sd_{epi}(epi)$
  Run one episode using $\mathcal{S}^a$
  Store all global history experience tuples $\boldsymbol{e}^{\boldsymbol{\tau}}_t = \langle s_t, z_t, \boldsymbol{\tau}_t, \boldsymbol{u}_t, r_t, s_{t+1}, z_{t+1}, \boldsymbol{\tau}_{t+1} \rangle$ in D
  Sample random minibatch B_{pro}, B_{atk} from D_{pro}, D_{atk}
  $\mathcal{L}_{imit}, \mathcal{L}_{dis}, \mathcal{L}_{class} \leftarrow$ Set to 0

  if $Sd_{bir}(epi)$ **then**
    $z, \tau \leftarrow$ take observations and observation histories from B_{atk}
    $u_{pro}, u_{atk} \leftarrow$ action selection according to $\pi_{pro}(z), \pi_{atk}(z)$
    $v_{pro}, v_{atk} \leftarrow$ discriminator input according to $(u_{pro}, \tau), (u_{atk}, \tau)$
    $\mathcal{L}_{dis} \leftarrow H_p(v_{pro}, v_{atk})$ w.r.t. Eq. 5.1
    $\mathcal{L}_{imit} \leftarrow H_{bs,p}(v_{pro}, v_{atk})$ w.r.t. Eq. 5.1 and followed by Eq. 5.2
  end if

  $r_F \leftarrow$ update all rewards r in B_{atk} using classifier w.r.t. Eq. 5.3
  $\mathcal{L}_{ant} \leftarrow \mathcal{L}(\theta_{atk})$ w.r.t. Eq. 2.1 with $Q^{\boldsymbol{\pi}}(\boldsymbol{\tau}_t, \boldsymbol{u}_t; \theta) = -Q^{\boldsymbol{\pi}}(\boldsymbol{\tau}_t, \boldsymbol{u}_t; \theta)$, B_{atk} and $r = r_F$

  $\mathcal{L}_{atk} \leftarrow \mathcal{L}_{ant} + \mathcal{L}_{imit}$

  if $Sd_f(epi)$ **then**
    $s'_{pro}, s'_{atk} \leftarrow$ select state features from B_{pro}, B_{atk}
    $\mathcal{L}_{class} \leftarrow H_p(s'_{pro}, s'_{atk})$ w.r.t. Eq. 2.12
  end if

  Update θ_{dis} by minimizing $\mathcal{L}_{dis}$
  Update θ_{class} by minimizing $\mathcal{L}_{class}$
  Update θ_{atk} by minimizing $\mathcal{L}_{atk}$

end for

6. Experiments

6.1. Experimental Details

For our experiments we require different ways of approaching each environment, since the balancing between hiding and sabotaging of the ISAAC Invader heavily depends on the complexity of the particular setting 5.3. Furthermore the Multi-Objective 2.6 of the ISAAC Invader poses a major challenge, because conventional single evaluation techniques are not sufficient to measure the success of our approach with regard to all objectives. For this purpose, we will evaluate several success indicators for each of the objectives and use them to measure the effectiveness of ISAAC. Therefore we compare among other metrics the positional progression of the ISAAC Invader throughout the entire episodes, measure the action selection of the substituted protagonist against those of the attacker, and conduct a survey with respect to the ability of hiding the attacker. In the following subsections we give a more detailed description of how we approach an environment, as well as provide information on which hyperparameters and architectures were used to train the agents.

6.1.1. Approaching an environment

The complexity of the environment has the greatest influence on the hiding ability of the ISAAC Invader. In particular, one has to take the positioning of the protagonists during the course of the episodes into account. Depending on the setting, there might exist very complex movement sequences or maneuvers which are significantly more difficult for the ISAAC Invader to imitate. On the other hand, if the target protagonist barely performs any movements, it may be possible for the ISAAC Invader to almost neglect the State Feature Imitation component which is defined in 5.2.2. Since the other state features, such as the health or shields, indirectly reflect the performance of the protagonists, we will only take the locations of all the agents for our state feature selection 5.3 in this work.

Another factor that influences the training of the ISAAC Invader is the relationship between the actions $\boldsymbol{u}$ and rewards R. Depending to what extent and whether an action contributes directly or indirectly to the overall reward, this will potentially have an impact on our equation in 5.1. Therefore, it is necessary to tailor the Behavior Imitation Training Scheduling Sd_{bir} in order to provide a stable and adequate training process, based on the environment. Besides the Sd_{bir} we also have to tune our other schedules to balance the imitation and the antagonistic parts. Nevertheless, in our experiments we will mostly only choose the same values in order to analyze the extent to which the various algorithms are more susceptible to our approach and to show what kind of problems we encounter when choosing the same parameters.

For the target of our attack, we choose protagonists that are particularly interesting for us, and we substitute them with an attacker. In the case of a homogeneous team regarding

the units, we simply select an arbitrary unit, whereas in the case of a heterogeneous one, we pick a specific unit that will be the victim of our compromise.

6.1.2. Hyperparameters and Architectures

Before we can start to apply our approach, we first need trained protagonists. For this purpose, we train agents in their respective environments by making use of the algorithms which were described in 2.2: QTRAN, QMIX, QPLEX, VDN, IQL and OW-QMIX. For exploration and exploitation 5.2.2, we employ the ε-Greedy Method [SB98], which we start from 1.0 and linearly decay over the course of $1e+5$ time steps until we reach 0.05. In total the training of the protagonists in each environment will have a duration of $2e+6$ time steps. In the process, at the end of each episode, we sample a training mini-batch with a size of 16 randomly from the replay memory of the agents which has a potential capacity of 5000 mini-batches. Subsequently on the basis of this sampled mini-batch, we train the RNN agents 2.2 which use the Root Mean Square Propagation optimizer (RMSprop) [Rud16] with $\alpha_{RMSprop} = 0.99$ and $\varepsilon_{RMSprop} = 1e-5$. The learning rate of the agents α are set to $5e-4$ and the interval at which we update the parameters for the target networks θ^- is 200. We use a discount factor γ of 0.99 for 2.1 to incorporate rewards further into the future for our current expected return. For the network architecture of the agents, we decided on an RNN agent, also DRQN (see 2.2), in which we make use of a Gated Recurrent Unit (GRU) [CvMBB14]. These networks are composed of a fully connected layer that maps the input onto 64 hidden features, which is followed by a 64 to 64 GRU, and finally again by a fully connected layer which decreases the 64 features onto the action space. For the OW-QMIX algorithm we set the α_W in 2.2 to $\alpha_W = 0.1$. In order to prevent exploding gradients during the training process, we clip the gradients above the L2 norm of 10.

Once we have completed the training of the protagonists, we keep the parameters of the networks θ fixed and do not continue to train them. Afterwards the training of the ISAAC Invader takes over, where it substitutes one of the trained protagonists. Throughout this thesis we restrict ourselves to only one protagonist to be replaced, although our approach can also be applied to any number of protagonists at the same time. As a matter of fairness we will keep the identical values and algorithms that were previously used for the protagonist training. However, the training of the ISAAC Invader is not performed until a time step limit is reached but rather for a given number of episodes. This allows us to better analyze the sabotage of the attacker throughout the course of the training, as our approach does not primarily focus on the individual rewards and is instead more centered around the win rates. The value of the threshold T_{bs} in 5.2 will be $T_{bs} = 1.0$, since this threshold lies in between the rewards during an episode and the win reward [SRdW$^+$19]. This should contribute towards reducing the win rate of the agents while simultaneously retaining the individual rewards throughout each episode. Regarding the classifier reward F in 5.3, we use a constant value of $F = 0.1$ since this is high enough to encourage the ISAAC Invader to reproduce the state features and at the same time is small enough that it does not cause other components to be neglected.

We choose the Leaky Rectified Linear Unit (Leaky ReLU) [HZRS15] as activation functions in the discriminator and classifier with a slope of $a = 0.25$ to counteract possible sparse gradients. The architecture of the discriminator consists of two input network segments, where we concatenate the outputs afterwards and pass the result into a fully

connected network section. For the input networks we also make use of fully connected networks, with one being responsible for the actions u and the other for the observation histories τ. The action input component consists of one 64 and one 32 hidden layer. As for the observation history input module, we first use one 512 and subsequently two 256 hidden layers. Afterwards the two outputs of the components will be concatenated and inserted into a fully connected network with the following hidden layers: 256, 256 and 64. The resulting value will then be transformed into a single float and mapped into the interval $(0, 1)$ with the help of the Sigmoid function [NIGM18], which yields us the predicted probability $p(y_v)$ from the discriminator for the action history pair $v = (u, \tau)$.

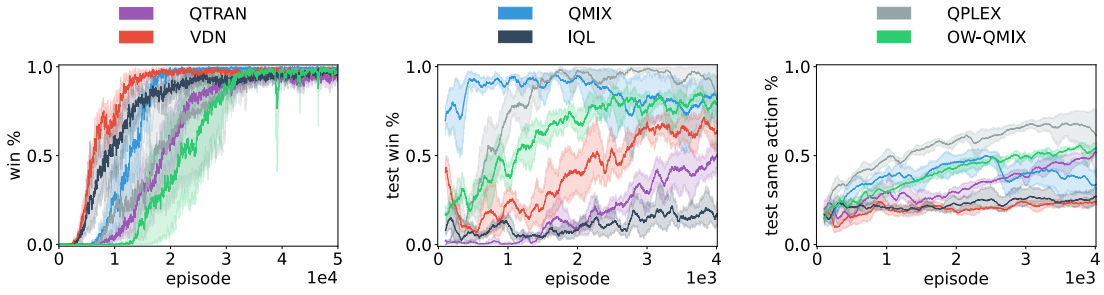
For the state features classifier, we use a singular fully connected network, where the hidden layers will be 1024, 512, 256 and 64. In the same way as with the discriminator, the output is then mapped with the Sigmoid function to a value between $(0, 1)$, which provides us with the predicted probability $p(y_{s'})$ from the classifier regarding the state features s'.

In the following sections, we are going to apply and evaluate our approach on three different scenarios taken from the SMAC.

6.2. 3 Stalkers vs 3 Zealots

The teams in the 3 Stalkers vs 3 Zealots scenario are each composed of homogeneous units as described in 4.1.4. Initially we train the agents who are controlling the Stalkers with no attacker in order to simulate a regular finished training and deployment, which is shown in 6.1a. Afterwards one of the trained protagonists is substituted with our ISAAC Invader and we train it over the course of 40000 episodes. During this process all of the protagonists select their respective actions solely based on the greedy action selection. For the training procedure of the ISAAC Invader we choose the Behavior Imitation Training Scheduling Sd_{bir} of 5 to 1, the Mimic Feature Training Scheduling Sd_f of 5 to 1, as well as the ISAAC Invader Episode Scheduling Sd_{epi} of 99 to 1. In order to benchmark the success performance of our approach in comparison with other potential attack scenarios which make use of the very same attack vector, we additionally employ an agent who chooses random actions (Random Invader) as well as an obvious attacker (Naive Invader) who only pursues the objective described in 5.1. All of the training curves, both Test and Train, have been performed using 5 independent runs, taking the mean over 100 episodes and averaging over them. Furthermore we executed 1000 evaluation or test episodes for each of these independent runs after the training process has been completed, with the results being listed in the tables 6.1 and 6.2.

The training curves of the protagonists without any attackers are shown in 6.1a. We can observe that all agents successfully solve the environment 3 Stalkers vs 3 Zealots with all algorithms and QMIX, QPLEX, VDN, IQL as well as OW-QMIX achieve an average win rate of at least 0.96. Only the mean win rate of the QTRAN algorithm in this scenario lies at 0.82, which can be found in table 6.1. During the training process of the attackers, we performed in addition to the regular training episodes after every 10th episode a test episode in which all of the agents, both the protagonists and the attackers, choose their actions greedy. The resulting test win rates as well as the test counts of the same action selections made by protagonists and the hidden attackers are shown in 6.1b and 6.1c. The curves in each of the plots represent the mean values and the corresponding colorized regions indicate the 95% confidence intervals. We can quickly



(a) Protagonist Training Win Rates

(b) ISAAC Invader Training Test Win Rates

(c) ISAAC Invader Training Test Action Matching

Figure 6.1.: Learning curves of 5 independent runs in 3s vs 3z of various algorithms. Average (test) win and action matching rates with 95% confidence intervals: ISAAC decreases the test win rates (b) of the protagonists (a) while imitating the actions (c).

notice from the plot 6.1b that all the learning algorithms presented in this paper are not sufficiently resilient against the Adversarial Attacks which utilize agents as the attack vector. At the very beginning of the training phase of the ISAAC Invader, it becomes apparent that the test win rate of all algorithms is severely degraded. At this stage of the training the ISAAC Invaders have not yet been trained at all and perform random actions during their exploration phase. The fact that already a Random Invader can considerably reduce the overall performance of all other agents in the environment is also reflected in the table 6.1. In particular the algorithms QPLEX, IQL, QTRAN and OW-QMIX suffer a substantial loss in the test win rates by 96%, 76%, 84% and 68%, respectively. But also the performances of the agents who were trained using VDN decrease by 46% and the ones with QMIX by 10% (exact mean values and 95% confidence intervals can be found in table 6.1). During the training course of the ISAAC Invader, we observe increasing test win rates across all of the algorithms, which is a result of the imitation of the protagonists' behavior. Nevertheless, by the end of the 40000 training episodes all agents show a remarkable drop in their win rates, as seen in 6.1b and 6.1. Especially the protagonists who were trained using IQL suffer from a major decline to a test win rate of 27%. Therefore it is evident that the hiding component within the ISAAC Invader still causes the performance of the protagonists to drop. Correspondingly we observe the highest test win rate decrease for IQL by 72%, followed with 32% for QTRAN, 24% for VDN, 19% for OW-QMIX, 18% for QMIX and only 16% for QPLEX. When only considering the test win rates, we would initially assume that here IQL suffers the most from our approach, however, when we include all of the other metrics, we can see the

3s_vs_3z	No Substitution	ISAAC Invader	Naive Invader	Random Invader
QTRAN	0.82 ± 0.01	0.56 ± 0.01	0.01 ± 0.00	0.13 ± 0.01
QMIX	1.00 ± 0.00	0.82 ± 0.01	0.68 ± 0.01	0.90 ± 0.01
QPLEX	1.00 ± 0.00	0.84 ± 0.01	0.00 ± 0.00	0.04 ± 0.01
VDN	0.99 ± 0.00	0.75 ± 0.01	0.20 ± 0.01	0.53 ± 0.01
IQL	0.96 ± 0.01	0.27 ± 0.01	0.07 ± 0.01	0.23 ± 0.01
OW-QMIX	0.98 ± 0.00	0.79 ± 0.01	0.21 ± 0.01	0.31 ± 0.01

Table 6.1.: Win rates across 5×1000 evaluation episodes after training in 3s vs 3z of different algorithms: All Invader succeed in decreasing the protagonists win rates.

3s_vs_3z	No Substitution	ISAAC Invader	Naive Invader	Random Invader
QTRAN	1.00 ± 0.00	0.56 ± 0.00	0.23 ± 0.00	0.16 ± 0.00
QMIX	1.00 ± 0.00	0.34 ± 0.00	0.08 ± 0.00	0.15 ± 0.00
QPLEX	1.00 ± 0.00	0.66 ± 0.00	0.16 ± 0.01	0.15 ± 0.00
VDN	1.00 ± 0.00	0.26 ± 0.00	0.20 ± 0.00	0.15 ± 0.00
IQL	1.00 ± 0.00	0.24 ± 0.00	0.12 ± 0.00	0.16 ± 0.00
OW-QMIX	1.00 ± 0.00	0.53 ± 0.00	0.10 ± 0.00	0.16 ± 0.00

Table 6.2.: Action matching rates across 5×1000 evaluation episodes after training in 3s vs 3z of different algorithms: ISAAC Invader significantly outperform Naive and Random Invader.

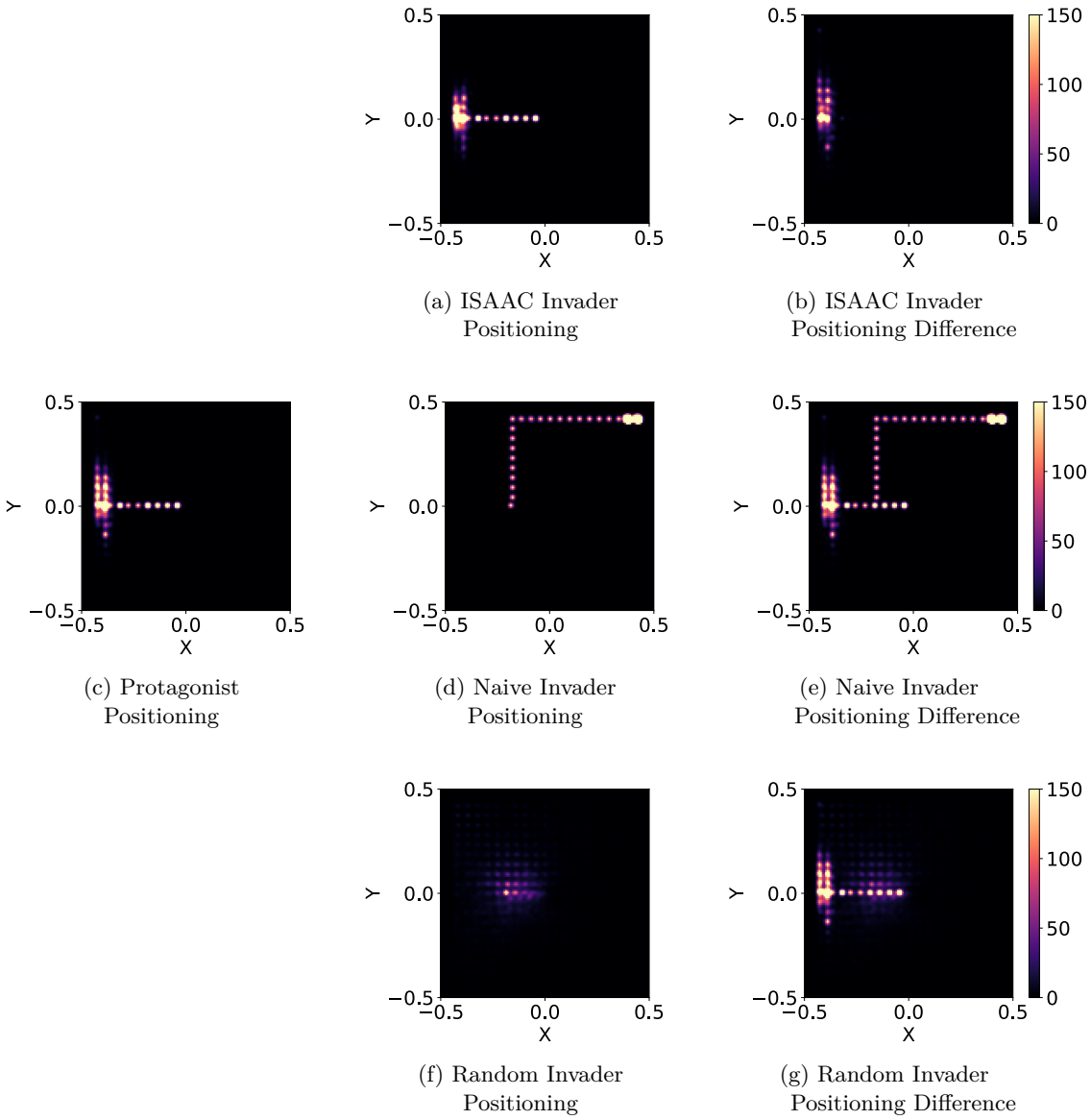
full picture. When training the Naive Invader, who does not try to hide his sabotage, we achieve a very drastic drop in the test win rate for almost all algorithms. The only algorithm that still maintains a reasonably high test win rate of 0.68 is QMIX, although it is considerably lower in comparison to the attack with the Random Invader 0.90 and the ISAAC Invader 0.82. The Naive Invader manages to achieve the largest reduction in test win rates for all algorithms.

6.2.1. Action Selection Matching

However, when examining the action choices of the attackers and protagonists in 6.2, it becomes apparent that both the Naive Invader as well as the Random Invader only select the same actions as the protagonists in a limited number of cases. Particularly the Random Invaders decide on the same action only in $15 - 16\%$ of the states, due to the limited action space consisting of no more than 6 possible actions. For the Naive Invader the highest match of actions lies around 0.23 for QTRAN, followed by VDN with 0.20. Despite the minimization of the joint action-value function Q^{π} of the protagonists, a higher action matching is found here in comparison to the Random Invader. Nevertheless, when it comes to action matching with the protagonists our ISAAC approach dwarfs both the Naive Invader as well as the Random Invader. Whereas VDN and IQL lie at 26% and 24% respectively, we reach a match of 66% for the QPLEX algorithm. Similarly we also observe a high action match with the other algorithms, with the rate for QTRAN being 56%, for OW-QMIX 53% and for QMIX 34%. These results 6.2 show that the imitation component of our approach is especially effective for QPLEX, QTRAN and OW-QMIX in this setting. Thus, in the example of QPLEX, our ISAAC Invader selects the very same action as the protagonist in 2 out of 3 states while simultaneously reducing the test win rate by 16%. Likewise we see in the case of QTRAN that with 56% same action choice, we can decrease the performance of the agents in this scenario by 32%. For the algorithms QMIX and OW-QMIX the match rates are 0.34 and 0.53 in this environment, which is also significantly higher when compared to the Naive Invader or the Random Invader. Therefore we have shown that our ISAAC approach is successful in decreasing the test win rates of the protagonists, while at the same time the ISAAC Invader is better at hiding compared to an Naive Invader or Random Invader in terms of the action selection.

6.2.2. Positioning of Attackers

When focusing on the positional sequences of the agents, we can observe a similar result as for the action choices. In 6.2 we have drawn the location histories for the algorithm QPLEX based on evaluation runs each containing 1000 episodes. Displayed here are the locations of the attackers of one evaluation run within the environment, which are represented in the form of a heat map. Thus 6.2c shows the heat map of the substituted protagonist, which serves as a reference for our attack scenarios. Upon closer inspection of the plots 6.2a, 6.2d and 6.2f, we can recognize that only our approach shares similarities with the substituted protagonist. In the case of the Random Invader, seen in 6.2f, the heat map appears to be a normal distribution centered around the starting position. For this reason it is very unlikely for the Random Invader to be capable of deceiving any external observer. By contrast, the Naive Invader follows a well-defined path into one of the corners of the environment and consequently avoids the combat all together.



(a) ISAAC Invader Positioning

(b) ISAAC Invader Positioning Difference

(c) Protagonist Positioning

(d) Naive Invader Positioning

(e) Naive Invader Positioning Difference

(f) Random Invader Positioning

(g) Random Invader Positioning Difference

Figure 6.2.: Positioning heat map of 1 out of 5 evaluation runs using QPLEX in 3s vs 3z each consisting of 1000 episodes: Only ISAAC manages to achieve similar positioning to the protagonists.

Here the heat map 6.2d indicates rather clearly that the Naive Invader keeps taking the same route over and over again which makes it easy to identify it by monitoring several episodes. Only ISAAC can reproduce a similar positional progression with respect to the substituted protagonist. The differences between the attackers and the original agent are displayed respectively in plots 6.2b, 6.2e and 6.2g. Here we observe a significant deviation of the Naive and Random Invaders from the protagonists with respect to the positioning. The heat map of the ISAAC Invader 6.2b reveals a particularly high similarity at the beginning of the episode, such that the difference in the regions around the starting

location is close to zero. As the episodes progress the ISAAC Invader remains close to the positions of the protagonist, however the intensities of both agents differ. This may be due to the length of the episode, since for the protagonists a successful episode tends to be of a rather short duration. Therefore the ISAAC Invader has to remain longer in the environment as a result of the longer episodes. Despite this, the shape of the mismatch still appears to be in strong accordance with the positions of the protagonists. When taking all 5 evaluation runs of QPLEX into account, the locations of the ISAAC Invader matches with $49.44 \pm 20.99\%$, the Naive Invader with $1.31 \pm 2.16\%$ and the Random Invader with $4.76 \pm 0.20\%$ the positions of the substituted protagonist. We can consequently deduce here that our introduced invader performs rather well in terms of hiding its locations within the environment. Furthermore, with our approach the ISAAC Invader outperforms the Naive Invader in this regard by a factor of 37 as well as the Random Invader by a factor of 10.

6.2.3. Summary

In the environment 3 Stalkers vs 3 Zealots we have demonstrated that our approach not only manages to significantly decrease the overall win rates of the agents 6.1, but also to remain stealthy with respect to the action selection 6.2 as well as the positioning 6.2 in the environment itself. Although our approach slightly diminishes the performance reduction in comparison to the Naive and Random Invader, it is much more effective at staying hidden as a result. Because we are dealing with a Multi Objective Optimization Problem 2.6, there is no need for the hiding of the attacker to be perfect. To what extent this trade-off has to be taken, will be investigated in the SMAC scenario 5 Marines vs 6 Marines (for 3s vs 3z also see A.1).

6.2.4. Notes and Discussion

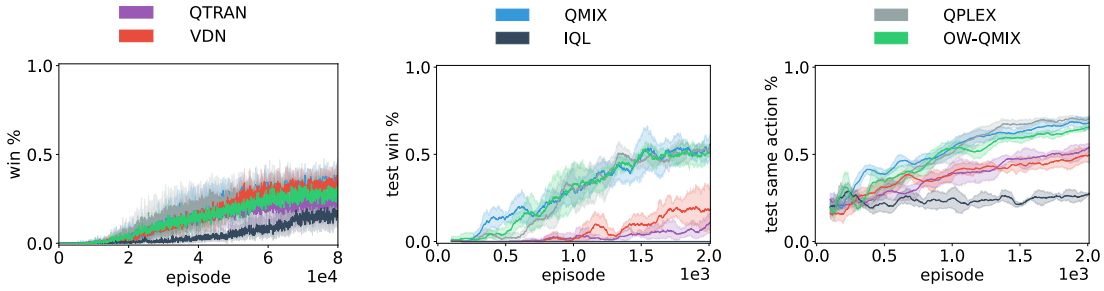
While the results in the tables 6.1, 6.2 and the plots 6.1b, 6.1c are already looking quite promising, there is still a lot of unused potential to be found here. By examining the loss of the ISAAC Invader A.4a, it becomes apparent that the chosen hyperparameters may have a very strong impact on the training success of the attacker. In the example of QMIX, the relatively low same action selection of the ISAAC Invader with respect to the substituted protagonist may be due to the fact that the antagonistic and imitation components have not been well balanced in this case. This is also reflected in A.4a, where the loss becomes increasingly higher beginning at the test episode of approximately 3000 and thus negatively affecting the training process. Therefore, the hyperparameters for each algorithm and environment, in particular the Behavior Imitation Training Scheduling Sd_{bir}, needs to be tweaked for an optimal training and can not set to the same fixed values as it was done in this experiment.

In order to measure the imitation objective in this work, we have decided to adopt a number of different metrics. Although the Kullback-Leibler divergence [KL51] or the Jensen-Shannon divergence [Lin91] are typically used in the evaluation of Generative Adversarial Networks, these can not be applied in this context. The reason for this lies in the Multi Objective, because the antagonistic part distorts the behavior of the ISAAC Invader making the difference between the distributions of the action selections of the protagonist and the ISAAC Invader no longer a meaningful metric. Additionally, the

exact underlying distribution of the protagonist's outputs is not directly given, but is only available in the form of a neural network model, which presents a major challenge when computing the distance. Similarly, based on the aforementioned issues, other techniques which directly measure the similarity between ML models [KNLH19] may also not be applicable here.

6.3. 5 Marines vs 6 Marines

The 5 Marines vs 6 Marines SMAC environment is homogeneous and asymmetric, for further information see 4.1.4. While the scenario is rated hard for regular agents, this does not necessarily apply to our attackers. Once the agents have been trained without an Adversarial Attack, we again substitute one protagonist by the attacker in each of the algorithms. Afterwards the learning process of the attackers is performed over the course of 20000 episodes. We also initially apply the same schedules here as we have done for the 3 Stalkers vs 3 Zealots scenario. The training curves of the protagonists as well as the ISAAC Invaders can be found in 6.3. Those of the Naive Invader are located in Appendix A.5b, but are of less interest for our evaluation. In the regular training of the agents in 6.3a and table 6.3, we observe that the protagonists do not achieve a higher test win rate of 80% even in the absence of attackers (note that the win rate and the test win rate may differ significantly). The subsequent deployment of an attacker substantially degrades the win rates of the agents, resulting in test win rates for the Naive Invader and Random Invader of 0.00 across all algorithms. Similar to what we have witnessed previously in the 3 Stalkers vs 3 Zealots environment, we can observe an increase in the test win rates and the number of same actions over the course of the training episodes in the curves of the ISAAC Invader 6.3b, 6.3c. We once again succeeded in reducing the test win rates considerably (see table 6.3 and 6.3b), while the match between the actions of the protagonist and those of the ISAAC Invader is rather high in the respective states. More specifically, the same action selection rates of QMIX and QPLEX are 69%, of OW-QMIX are 66%, of QTRAN are 51%, of VDN are 49% and of IQL are 25%. The respective test win rates 6.3 reveal a reduction of 26% in the case of QMIX, 13% for QPLEX, 19% for OW-QMIX, 85% for QTRAN, 68% for VDN and 100% for IQL.



(a) Protagonist Training Win Rates

(b) ISAAC Invader Training Test Win Rates

(c) ISAAC Invader Training Test Action Matching

Figure 6.3.: Learning curves of 5 independent runs in 5m vs 6m of various algorithms. Average (test) win and action matching rates with 95% confidence intervals: During training some test win rates of ISAAC (b) exceed the win rates of the protagonists (a) due to high action matching rates (c).

6.3.1. Survey on hiding

Such a high degree of action matching raises the question of how much the ISAAC Invader has to be hidden in order for it to be indistinguishable from the actual protagonists in

5m_vs_6m	No Substitution	ISAAC Invader	Naive Invader	Random Invader
QTRAN	0.61 ± 0.01	0.09 ± 0.01	0.00 ± 0.00	0.00 ± 0.00
QMIX	0.76 ± 0.01	0.56 ± 0.01	0.00 ± 0.00	0.00 ± 0.00
QPLEX	0.62 ± 0.01	0.54 ± 0.01	0.00 ± 0.00	0.00 ± 0.00
VDN	0.72 ± 0.01	0.23 ± 0.01	0.00 ± 0.00	0.00 ± 0.00
IQL	0.57 ± 0.01	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
OW-QMIX	0.70 ± 0.01	0.57 ± 0.01	0.00 ± 0.00	0.00 ± 0.00

Table 6.3.: Win rates across 5×1000 evaluation episodes after training in 5m vs 6m of different algorithms: ISAAC decreases the win rates while Naive and Random drop them to zero.

5m_vs_6m	No Substitution	ISAAC Invader	Naive Invader	Random Invader
QTRAN	1.00 ± 0.00	0.51 ± 0.00	0.09 ± 0.00	0.18 ± 0.00
QMIX	1.00 ± 0.00	0.69 ± 0.00	0.04 ± 0.00	0.16 ± 0.00
QPLEX	1.00 ± 0.00	0.69 ± 0.00	0.14 ± 0.00	0.18 ± 0.00
VDN	1.00 ± 0.00	0.49 ± 0.00	0.02 ± 0.00	0.18 ± 0.00
IQL	1.00 ± 0.00	0.25 ± 0.00	0.08 ± 0.00	0.17 ± 0.00
OW-QMIX	1.00 ± 0.00	0.66 ± 0.00	0.12 ± 0.00	0.18 ± 0.00

Table 6.4.: Action matching rates across 5×1000 evaluation episodes after training in 5m vs 6m of different algorithms: ISAAC Invader significantly outperform Naive and Random Invader.

the eyes of an external viewer. Since we are dealing with a Multi Objective Optimization Problem 2.6, we are interested in finding a trade-off between the performance degradation of the agents 5.1 and the stealthing of the ISAAC Invader 5.2. In order to determine a specific threshold above which it is no longer possible for an outside observer to tell the difference between the attackers and the protagonists, we conducted a survey. For this purpose the participants were shown several video replays, each representing a single episode containing a different attack scenario. Afterwards the volunteers were asked to make a statement as to whether they perceived at least one attacker within this replay and accordingly they had to identify all of the attackers in the environment. The results of this survey covering the 5m vs 6m environment using QMIX are provided in 6.5, in which a total number of 22 subjects contributed. For this purpose, besides the Naive, No and Random Invader, we made use of ISAAC Invaders with different levels of action matching. When looking at the results of the replay without any attacker 6.5 (first column), we can see that 10 out of 22 participants have perceived at least one attacker. For the Random, Naive, and ISAAC Invader with 0.29 attack variants, we observe a statistically significantly higher degree of attacker detection by the subjects. Accordingly all 22 of the participants noticed at least one attacker in the Random Invader replay. In the case of the ISAAC Invader with 0.29, this number amounted to 21 and for the Naive Invader it was 16 subjects. By inspection of the True Negative rates in 6.5, we can observe that the attackers in the Random Invader case were identified 95%, those in the Naive Invader case 85%, and the ones in the ISAAC Invader 0.29 scenario 100% correctly. Likewise, when looking at the misclassified protagonists, we see a statistically

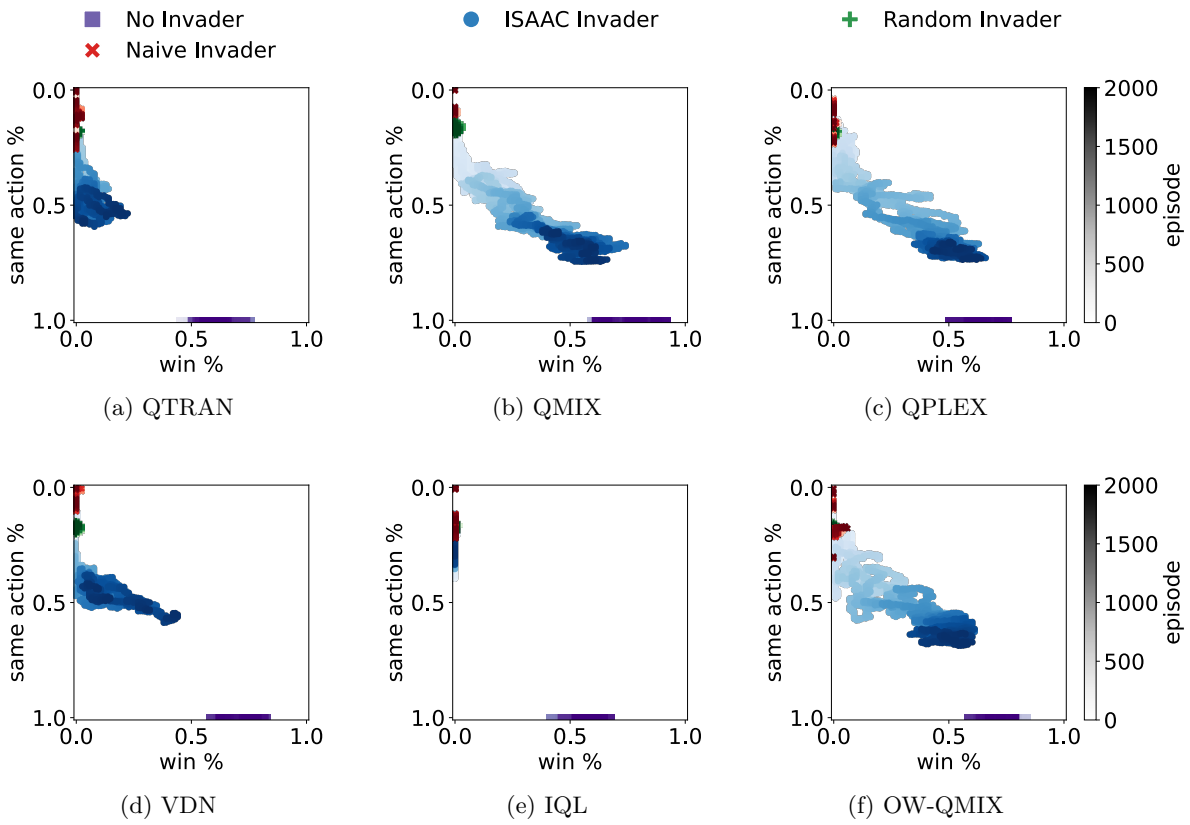
Survey 22 P | 5m_vs_6m	No	Random	Naive	ISAAC	ISAAC	ISAAC	ISAAC	ISAAC
Same action rate (mean)	1.00	0.16	0.04	0.29	0.45	0.54	0.59	0.69
Winrate (mean)	0.76	0.00	0.00	0.02	0.14	0.28	0.35	0.56
Invader id. present	10	22	16	21	6	9	11	16
Correct invader id.	0	21	13	21	2	2	3	4
Protagonists as invader id.	21	4	7	6	10	16	14	30
Pro. as invader id. per ag.	4.2	1	1.75	1.5	2.5	4	3.5	7.5
True Positive Rate	0.58	0.95	0.89	0.93	0.58	0.56	0.68	0.53
True Negative Rate	–	**0.95**	**0.81**	**1.00**	**0.33**	**0.22**	**0.27**	**0.25**
False Negative Rate	0.42	0.05	0.11	0.07	0.42	0.44	0.32	0.47
False Positive Rate	–	**0.05**	**0.19**	**0.00**	**0.67**	**0.78**	**0.73**	**0.75**

Table 6.5.: Survey regarding the hiding of attackers in 5m vs 6m using QMIX with 22 participants.

comparatively small amount of 4, 7, and 6 in the respective cases. Therefore all of the three invaders can be recognized rather easily in the replays and are not sufficiently hidden. In contrast, there were fewer attackers perceived during replays involving our ISAAC Invaders, which had action matches of 45%, 54%, 59% and 69%, respectively. For the ISAAC Invader instance with 45%, only 6, with 54% 9, with 59% 11, and with 69% 16 times it was reported that there is at least one attacker within the scenario. Also the corresponding True Negative rates are comparatively low amounting in all cases to a rate below 0.34, see 6.5. Examining the False Negative rates, we observe that these are higher compared to the True Negative rates. This suggests that the ISAAC Invaders are statistically well hidden in these shown replays. The False Positive rates in particular support this claim, as they are 67% for the ISAAC Invader with 45%, 78% for 54%, 73% for 59% and 75% for 69%. This survey hints that an action matching rate of 45% is sufficient to deceive external human observers in this particularly SMAC environment.

6.3.2. Trade-off between compromising and hiding

We next examine the trade-off between the same action selections and overall performance mitigation by the ISAAC Invaders during the training in the environment. For this purpose we make use of the test episodes during the training process in order to evaluate the potential application of the ISAAC Invader. The trade-off of the 5m vs 6m scenario over the training duration for each of the algorithms for 5 individual runs is shown in 6.4. Additionally the Naive and Random Invaders as well as the protagonist are also included as a reference. Concerning our approach we aim to achieve the lowest possible test win rate while simultaneously achieving a high action matching rate. This is why we invert the same action axis allowing us to minimize the two objectives in the context of a Multi Objective Optimization Problem 2.14. Consequently, it is desirable to have a shorter distance of the curve from the origin of the coordinates when looking at the plots in 6.4. Whereas the training curves of the Random and Naive Invaders are rather stationary here, we see a progression in the context of the trade-off for the ISAAC Invaders. At first we can observe that the test win rates tend to increase slowly as the numbers of same action choices grow. After a certain threshold, for every algorithm, environment and trained



(a) QTRAN (b) QMIX (c) QPLEX

(d) VDN (e) IQL (f) OW-QMIX

Figure 6.4.: Learning curves of 5 independent runs in 5m vs 6m of various algorithms. Trade-off between the test action matching and the test win rates.

protagonist different, the test win rate climbs significantly more quickly afterwards. Based on the curves of the ISAAC Invader 6.4, we can therefore tell how effective our approach is on each of the respective algorithms. Especially for the algorithms QMIX 6.4b, QPLEX 6.4c and OW-QMIX 6.4f the ISAAC Invader is able to mimic the actions relatively well. However, when incorporating the conclusion from the prior survey that a matching rate of 45% suffices for hiding, we are also able to successfully hide our approach when using the QTRAN 6.4a and VDN 6.4d algorithms. Only with IQL 6.4e does the ISAAC Invader fail to be successful at mimicking the actions of the protagonist. If a lower or higher rate of stealth is desired, the corresponding test win rates can also be found in 6.4. Indeed, in all algorithms we observe that the test win rates are significantly lower compared to the finished trained ISAAC Invaders with an action matching rate of 45%. This can also be seen in table 6.5, where the mean test win rates and action matching rates for the QMIX case from the survey have been recorded. According to the results, the ISAAC Invader remains hidden and is able to decrease the overall test win rate from 76% to 14%, which corresponds to a performance degradation rate of 81.58%.

6.3.3. Summary

Thus the success of our approach is also evident in the 5 Marines vs 6 Marines environment. It is possible to achieve a significant drop in the test win rates for all of the

algorithms tested in this work 6.3, while the same action selections are considerably higher than those of the Naive and Random Invader 6.4, except for the algorithm IQL. With respect to external observers, we manage to hide the ISAAC Invader in the case of QMIX starting from an action matching rate of 45%, see 6.5. Therefore we achieve the primary goal of our approach in this work for all algorithms except IQL, which is to hide our ISAAC Invader from external observers.

6.3.4. Notes and Discussion

Generally it appears to be easier for the ISAAC Invader to learn to imitate in the 5 Marines vs 6 Marines environment 6.3c compared to the 3 Stalkers vs 3 Zealots scenario 6.1c. One reason for this could be related to the complexity of the positioning of the agents. Whereas in 3s vs 3z the agents have to execute considerably more movement actions to relocate, in 5m vs 6m comparatively only little attention is given to the precise placement of the agents. Furthermore, the total number of agents may also impact the learning success of the ISAAC Invader. Provided that the success of the environment is less dependent on the individual agents, the impact of a single attacker in terms of performance mitigation will be smaller as well. In order to be able to compare the settings 3s vs 3z and 5m vs 6m with each other properly, the schedules in both environments were chosen to be the same. However, we have also conducted other experiments to study the influence of the schedules on the training process of the agents, the results of which can be found in A.3.

With respect to the survey in 6.5, it is important to point out that the results were obtained from only 22 participants. Consequently such a small sample is only significant to some extent and is therefore not representative of the general population. Furthermore the replays have been hand-selected from a volume of 5 episodes each and may not represent the average episode in the environment. Additionally most of the volunteers have a background in computer science, which may introduce a further bias. The participants were also instructed to actively search for attackers, which may lead to a expectation bias [WPKD12] in the perception of the observers and thus to a bias in the results. This is reflected in the high attacker detection rate of the replay with no invader, in which 10 out of 22 participants claimed to have perceived at least one attacker.

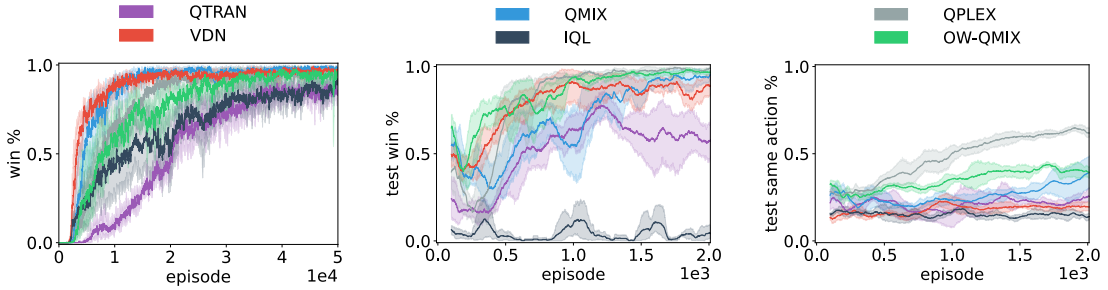
Regarding the plots in 6.4, these merely display the action matching rates in relation to the test win rates. While the same action selections of the attackers do provide a measurement of some part of the hiding, they do not reflect the entirety of it. This means that it is possible for the action matching rate to be very high and yet for the attacker to be easily identified in the environment. Thus the deductions, based on it, can only be drawn on other algorithms in 6.3.2 with a grain of salt. However, the underlying reason why we still choose to do so in our work is due to the lack of an all-encompassing metric for measuring stealth and the observations of the replays.

As a consequence, it is very challenging to find an optimal trade-off for the ISAAC Invaders, since for each environment and algorithm we require a different set of hyperparameters and in addition have to conduct a survey with a statistically significant sample size in order to verify the success of our approach in the environment.

6.4. 1 Medivac, 2 Marauders & 7 Marines

The third scenario we want to test our approach in is the 1 Medivac, 2 Marauders & 7 Marines environment (see 4.1.4). While in all the other scenarios every agent is able to actively contribute to the performance by dealing damage, in this setting a healing unit exists. This unit can only support and heal its friendly ones, and therefore it can not receive rewards actively by performing actions in the environment. For this scenario, we again initially trained the protagonists with no attackers. The corresponding training episodes can be seen in 6.5a while the resulting test win rates of the algorithms after the training can be found in table 6.6. Subsequently we train our ISAAC Invader as well as the Naive and Random Invader over the course of 20000 episodes, whose test training progress is shown in 6.5b and 6.5c. For this purpose we substitute the Healer Unit in the scenario with the attacker in order to evaluate our approach with respect to an agent with indirect influence. Afterwards, just like the previous scenarios, we conducted 1000 test episodes for each of them, the results of which are reflected in the tables 6.6 and 6.7.

Here we can find the test win rates to be again rather low in the case of the Naive and Random Invader. Nevertheless, with the algorithm VDN the test win rate for the Naive Invader remains high at 54%. When taking a look at the ISAAC Invader, we can only observe a very small decrease in the test win rates of the algorithms with the exception of IQL 6.6. The decrease in the case of IQL is 90%, of QTRAN 27%, of QMIX 3%, of VDN 2%, of OW-QMIX 1% and of QPLEX 0%, respectively. When including the same action selection made by the ISAAC Invaders and the protagonists 6.7, it becomes apparent that in the case of IQL, the comparatively high amount of loss can be attributed to the poor imitation success. However, for all other algorithms it also appears that the ISAAC Invader tends to have difficulties in imitating the actions of the protagonist in this environment. Only in the case of QPLEX a higher action matching rate of 50% can be found. By examining the training trade-off curves of the attackers in 6.6, we can state that in this environment, even a slight matching of actions is sufficient for the test win rate to substantially increase. Beginning at approximately a 20% same action selection by the ISAAC Invader, the test win rates begin to improve significantly. When



(a) Protagonist Training Win Rates

(b) ISAAC Invader Training Test Win Rates

(c) ISAAC Invader Training Test Action Matching

Figure 6.5.: Learning curves of 5 independent runs in MMM of various algorithms. Average (test) win and action matching rates with 95% confidence intervals: Even for modest action matching (c), the test win rate of ISAAC (b) rapidly increases up to that of the protagonist (a).

MMM	No Substitution	ISAAC Invader	Naive Invader	Random Invader
QTRAN	0.83 ± 0.01	0.61 ± 0.01	0.23 ± 0.01	0.21 ± 0.01
QMIX	1.00 ± 0.00	0.97 ± 0.00	0.10 ± 0.01	0.12 ± 0.01
QPLEX	0.99 ± 0.00	0.99 ± 0.00	0.00 ± 0.00	0.01 ± 0.00
VDN	0.99 ± 0.00	0.97 ± 0.00	0.54 ± 0.01	0.47 ± 0.01
IQL	0.89 ± 0.01	0.09 ± 0.01	0.02 ± 0.00	0.29 ± 0.01
OW-QMIX	0.99 ± 0.00	0.98 ± 0.00	0.12 ± 0.01	0.15 ± 0.01

Table 6.6.: Win rates across 5×1000 evaluation episodes after training in MMM of different algorithms: ISAAC win rates are close to the protagonists while Naive and Random are able to reduce them.

MMM	No Substitution	ISAAC Invader	Naive Invader	Random Invader
QTRAN	1.00 ± 0.00	0.24 ± 0.00	0.07 ± 0.00	0.13 ± 0.00
QMIX	1.00 ± 0.00	0.41 ± 0.00	0.19 ± 0.01	0.12 ± 0.00
QPLEX	1.00 ± 0.00	0.61 ± 0.00	0.12 ± 0.00	0.13 ± 0.00
VDN	1.00 ± 0.00	0.21 ± 0.00	0.07 ± 0.00	0.12 ± 0.00
IQL	1.00 ± 0.00	0.11 ± 0.00	0.08 ± 0.00	0.10 ± 0.00
OW-QMIX	1.00 ± 0.00	0.39 ± 0.00	0.08 ± 0.00	0.12 ± 0.00

Table 6.7.: Action matching rates across 5×1000 evaluation episodes after training in 5m vs 6m of different algorithms: ISAAC Invader perform better than Naive and Random Invader, although to a lesser extent.

the action matching rate exceeds 40%, in most of the cases the test win rate tends to be extremely close to the scenario with no attacker, see 6.6c. As a result, in order to have a somewhat realistic attack, the same action selections have to be below 30%, making it easy to identify the ISAAC Invader (based on the previous survey in 5m vs 6m 6.5).

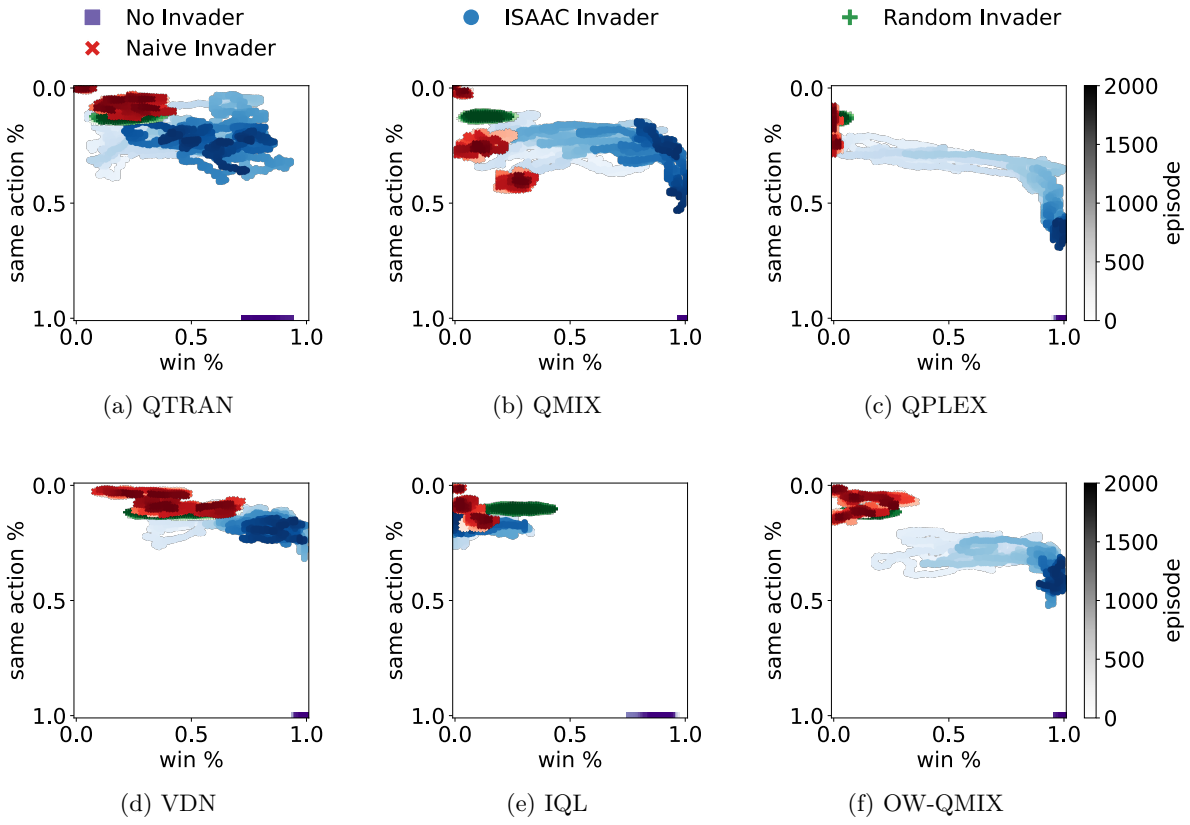
6.4.1. Summary

Because of the circumstances in this environment, it is extremely difficult for our attackers to impersonate the protagonist. Furthermore, in case of a successful imitation the test win rate becomes so high, as seen in the example of QPLEX 6.6 or 6.6c, that our attack can not really be considered to be effective at all. Therefore it is rather challenging to decrease the test win rates in this environment and simultaneously to remain well hidden.

6.4.2. Notes and Discussion

Part of the reason for the moderate success of our approach in this setting involves the indirect influence of the Healing Unit on the rewards. One consequence of this limitation is that we can no longer apply an optimal filter on the loss of 5.2, resulting in a distortion of the imitation of the substituted protagonist. In addition the ISAAC Invader is not given the ability to clearly distinguish between the antagonist and imitation actions, which further complicates the training process. Another possible explanation could be the inherent nature of the Medivac, as it with its healing action can only contribute to the success of the other agents by providing them with more health. As a consequence the

non-substituted protagonists have a longer life span and thus tend to be more likely to be victorious. Therefore the scenario 1 Medivac, 2 Marauders & 7 Marines demonstrates that there are also shortcomings with our approach, which can however be addressed in future work.



(a) QTRAN (b) QMIX (c) QPLEX

(d) VDN (e) IQL (f) OW-QMIX

Figure 6.6.: Learning curves of 5 independent runs in MMM of various algorithms. Trade-off between the test action matching and the test win rates.

7. Conclusion

In this work we presented ISAAC, an Adversarial Attack against Reinforcement Learning agents, which employs the agents themselves as the attack vector. Consequently it is possible to apply ISAAC to all kinds of cooperative multi-agent systems. In the process, our attacker, also referred to as ISAAC Invader, not only sabotages the other agents, but additionally has the ability to remain hidden from external observers. For this purpose our ISAAC Invader consists of two components: an antagonistic and an imitation one. While the antagonistic part tries to minimize the reward, the imitation portion focuses on mimicking the behavior as well as certain state features of the protagonists in the environment. This type of approach should be seen as increasingly important given the growing adoption of Artificial Intelligence in our daily lives and the rising demand for robust and resilient agents. In this context our approach demonstrates the potential shortcomings of learning algorithms and may as well be taken as a benchmark tool.

The fact that ISAAC is successful across a wide variety of algorithms has been confirmed by our experiments. Here we have considered 6 different algorithms: QTRAN, QMIX, QPLEX, VDN, IQL and OW-QMIX. Our approach has been tested and evaluated in 3 different scenarios, the 3 Stalkers vs 3 Zealots, the 5 Marines vs 6 Marines and the 1 Medivac, 2 Marauders & 7 Marines, which were all taken from the SMAC, a Dec-POMDP. We looked at the win rates of the agents, as well as other aspects, and found evidence that any kind of invader including ours is able to decrease the win rates significantly. Additionally, in order to demonstrate the ability to hide, we also examined the action choices of the attackers. Thereby we found our introduced approach to be substantially better at hiding compared to both the Naive and Random Invaders. Besides the action matching, we also focused on the positioning of the invaders themselves in the environment over the course of the episodes. In this process we were also able to conclude that our approach is considerably better at recreating the positions in the environment. Furthermore, rather than to limit ourselves to individual metrics, we conducted a survey to verify our success in achieving the goal of our work which is to hide the Adversarial Attack. This survey has also confirmed our results and revealed that 45% of action matches are already sufficient to ensure a proper hiding of our ISAAC Invader. This enabled us to take a closer look at the underlying Multi Objective Optimization Problem, where we had to find a trade-off between the minimization of the rewards and the ability to be hidden. We observed that with a 45% action matching rate, which is required for the ISAAC Invader to be hidden, we are able to achieve a reduction in the win rate of over 80%. It is however rather challenging to find a decent trade-off using our approach, since the hyperparameters have to be tailored to the algorithms and the respective environment in each particular case. Throughout our experiments, we have observed that the following algorithms, in descending order, are particularly susceptible to our approach: QPLEX, QMIX, OW-QMIX, QTRAN, VDN and IQL. In addition we encountered environments, such as MMM, which contain agents that are inherently resilient against these types of attacks. In the case of the algorithms, the learning procedure of the agents also play a

crucial part, which is the reason as to why our approach may not work well against all algorithms, similar to the case of IQL. However, the approach we have introduced in this thesis is designed in a fairly simplistic manner, allowing it to be improved in terms of its shortcomings.

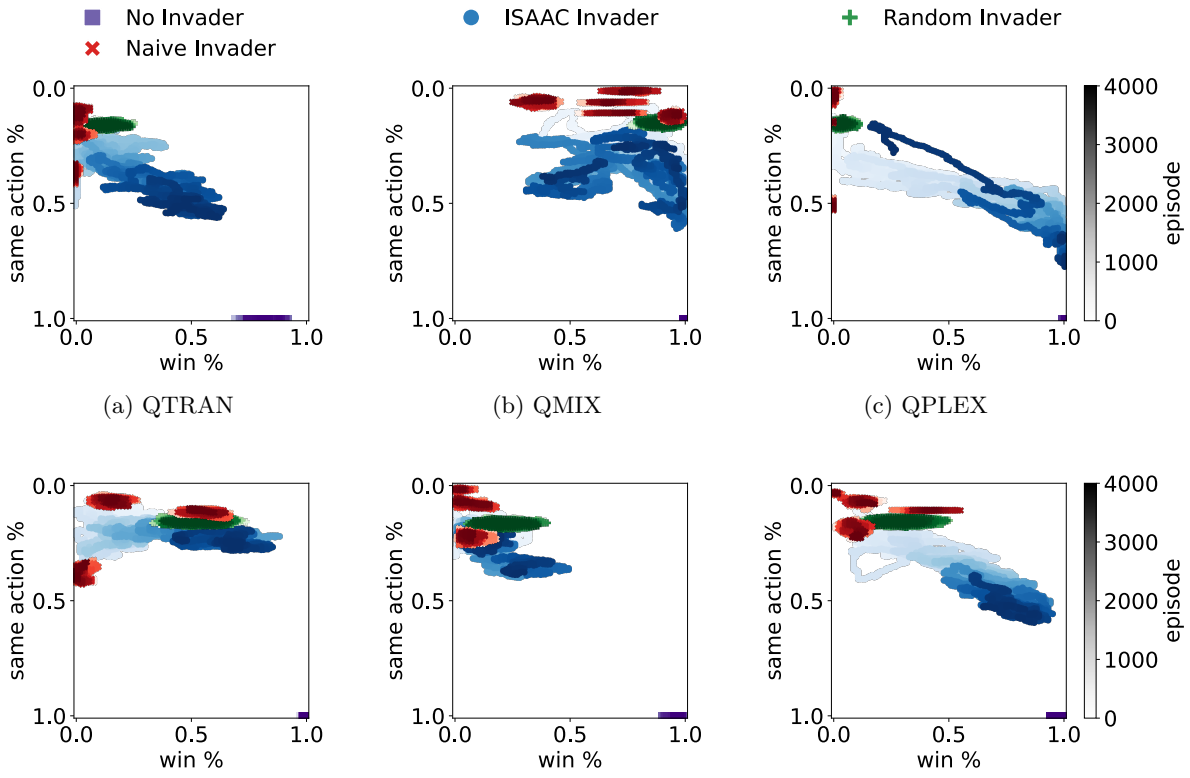
Ultimately, ISAAC has demonstrated that for now we can not fully rely on the usage of AI. In order to achieve this in the future, there is still a lot of improvement to be made with respect to the robustness and especially the resilience of Artificial Intelligence.

8. Future Work

ISAAC is, to our knowledge, the first Adversarial Attack designed to infiltrate the agents operating inside a multi-agent system with the purpose of sabotaging them while remaining hidden. In future work we intend to explore this approach in more detail with respect to the number of victim agents. While we only substituted one single protagonist in this work, it is possible to apply our approach to an arbitrary number of attackers. Especially with respect to other environments, it may be necessary to introduce multiple ISAAC Invaders in order to not only decrease the performance further, but also to better hide them through task sharing. In addition we will also target other Reinforcement Learning algorithms to benchmark them as well. In this process we will choose different learning algorithms for the protagonists and the attackers, thus enabling us to create an even more effective attack. Another aspect we will elaborate further on for our future work are the components of the ISAAC approach. Regarding the imitation part, we additionally are going to review different classifiers that may improve the learning process of hiding the attacker. Furthermore, since the Distance Measurement we chose to leverage in this thesis is the basic Binary Cross Entropy, we can consider substituting it with more sophisticated methods. This may additionally help to maintain a stable training process. While the primary focus in this work centered around the simultaneous hiding and the mitigation of the performance of the protagonists, in future work we intend to study the social aspects within the team in more detail as well. With respect to Game Theory, many interesting observations can be found with our approach, as already during our survey in the 5m vs 6m environment in this thesis we were able to discover a high False Negative rate in some cases, which may suggests that the protagonists are being used as scapegoats. Finally, we will explore potential defense mechanisms to counter these types of Adversarial Attacks in the future and thereby also contribute to the overall resilience of agents in multi-agent systems.

A. Appendix

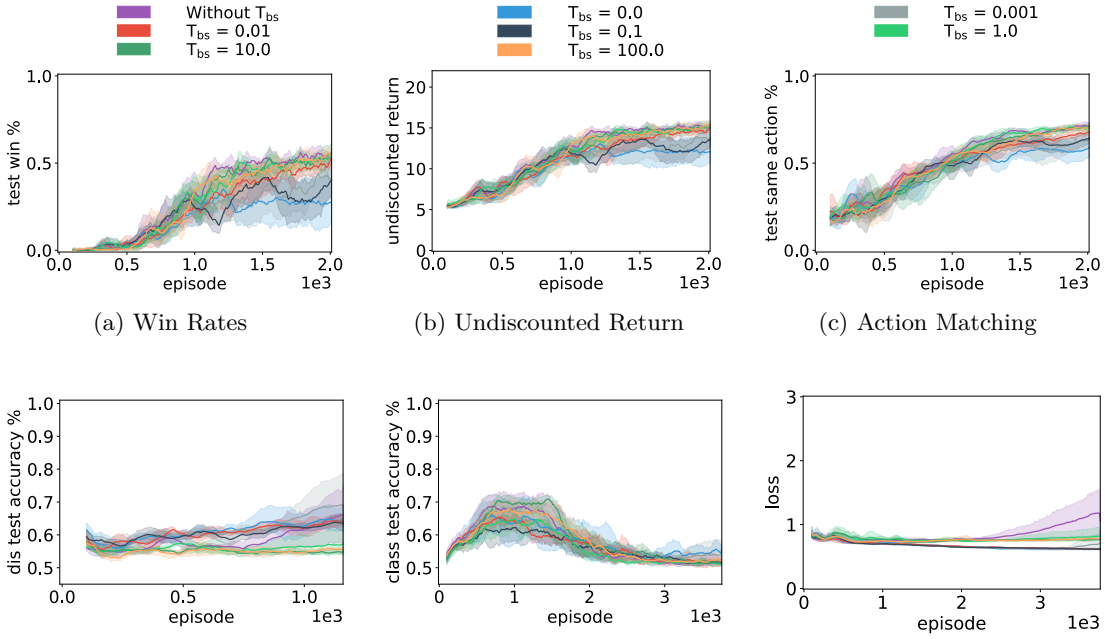
Omitted experiment plots



(a) QTRAN (b) QMIX (c) QPLEX

(d) VDN (e) IQL (f) OW-QMIX

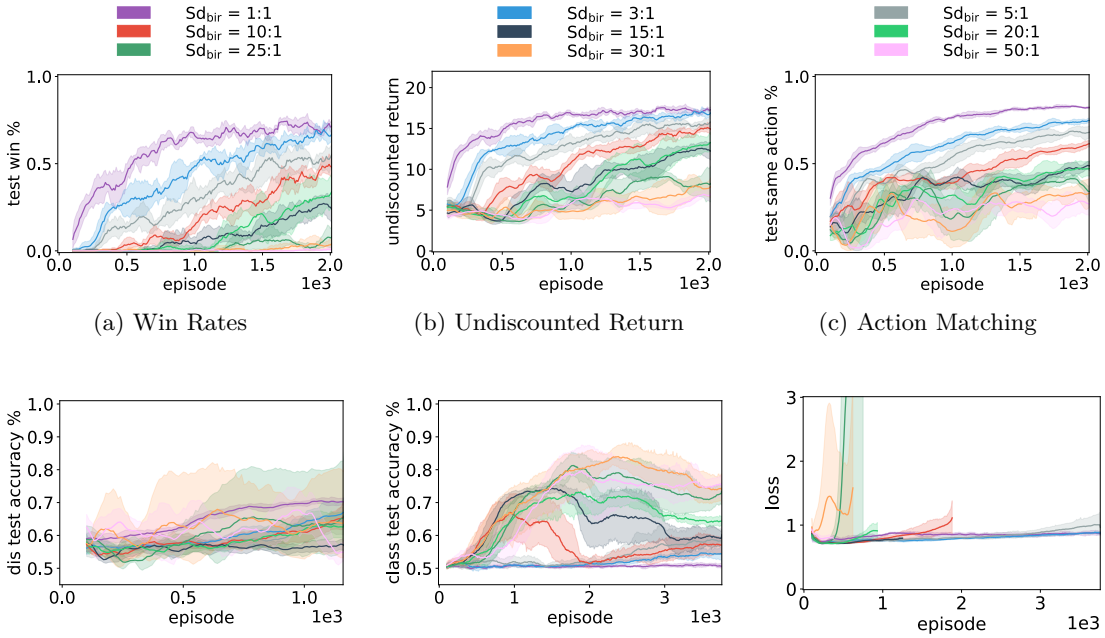
Figure A.1.: Learning curves of 5 independent runs in 3s vs 3z of various algorithms. Trade-off between the test action matching and the test win rates.



(a) Win Rates (b) Undiscounted Return (c) Action Matching

(d) Discriminator Accuracy (e) Classifier Accuracy (f) ISAAC Loss

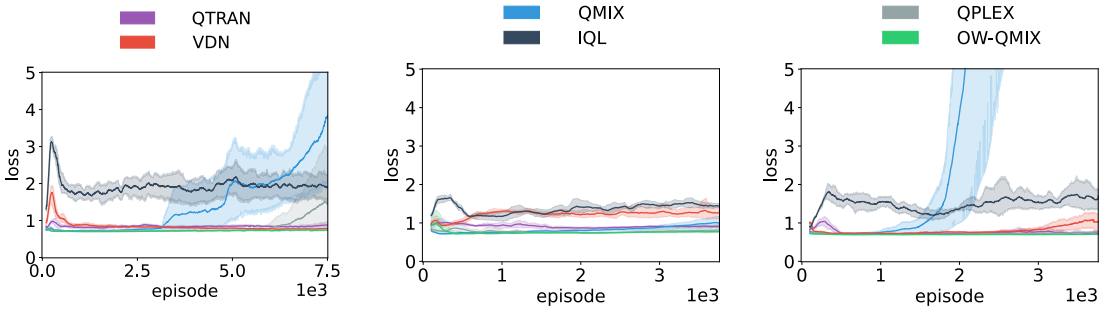
Figure A.2.: Learning curves of 5 independent runs in 5m vs 6m of QPLEX. Different values for the threshold T_{bs} in 5.2.1.



(a) Win Rates (b) Undiscounted Return (c) Action Matching

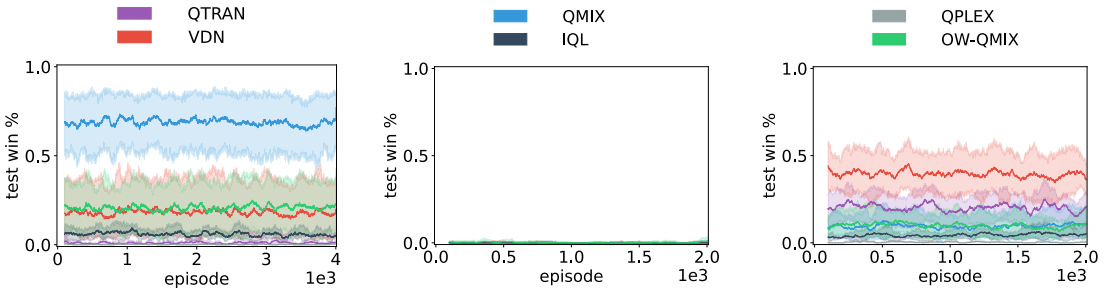
(d) Discriminator Accuracy (e) Classifier Accuracy (f) ISAAC Loss

Figure A.3.: Learning curves of 5 independent runs in 5m vs 6m of QMIX. Different behavior imitation ratios for the Behavior Imitation Training Scheduling Sd_{bir} in 5.3.



(a) 3s vs 3z ISAAC Loss (b) 5m vs 6m ISAAC Loss (c) MMM ISAAC Loss

Figure A.4.: Learning curves of 5 independent runs in different scenarios of various algorithms. Filtered Loss of the ISAAC Invader from the Behavior Imitation Process 5.2.1.



(a) 3s vs 3z Naive Invader Training Test Win Rates (b) 5m vs 6m Naive Invader Training Test Win Rates (c) MMM Naive Invader Training Test Win Rates

Figure A.5.: Learning curves of 5 independent runs in different scenarios of various algorithms. Average test win rates with 95% confidence intervals of the Naive Invader.

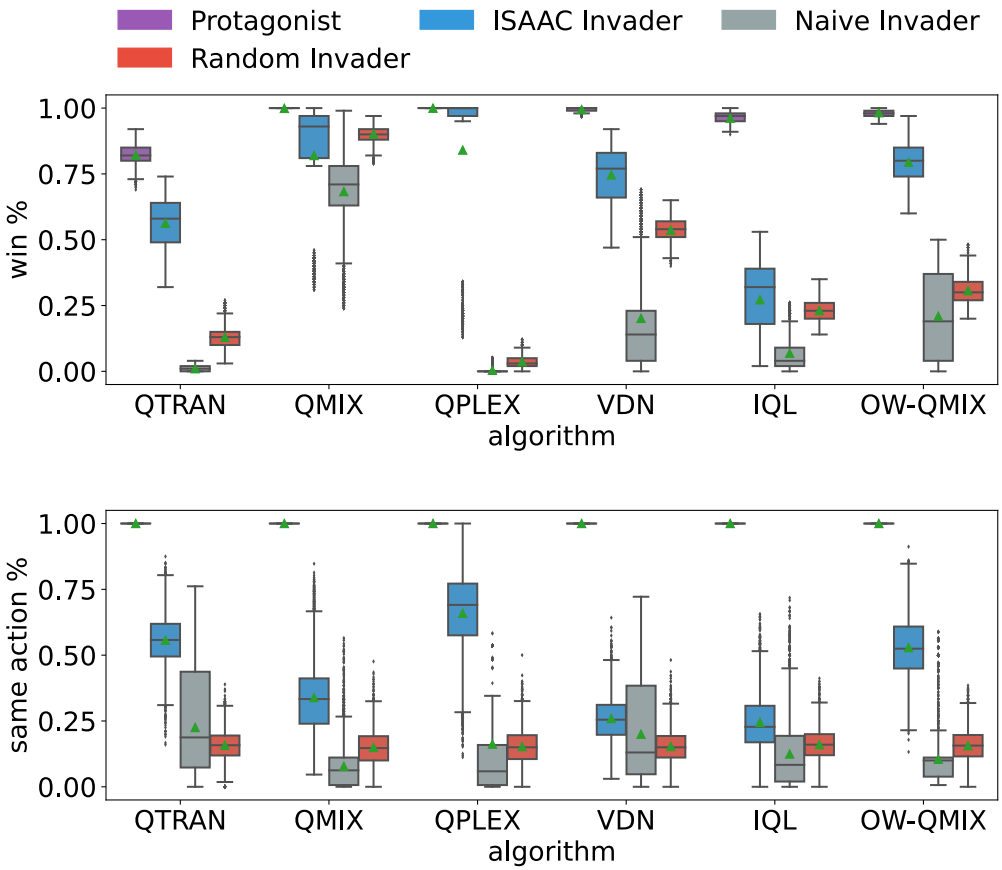


Figure A.6.: 5×1000 Evaluation episodes after training in 3s vs 3z of different algorithms.

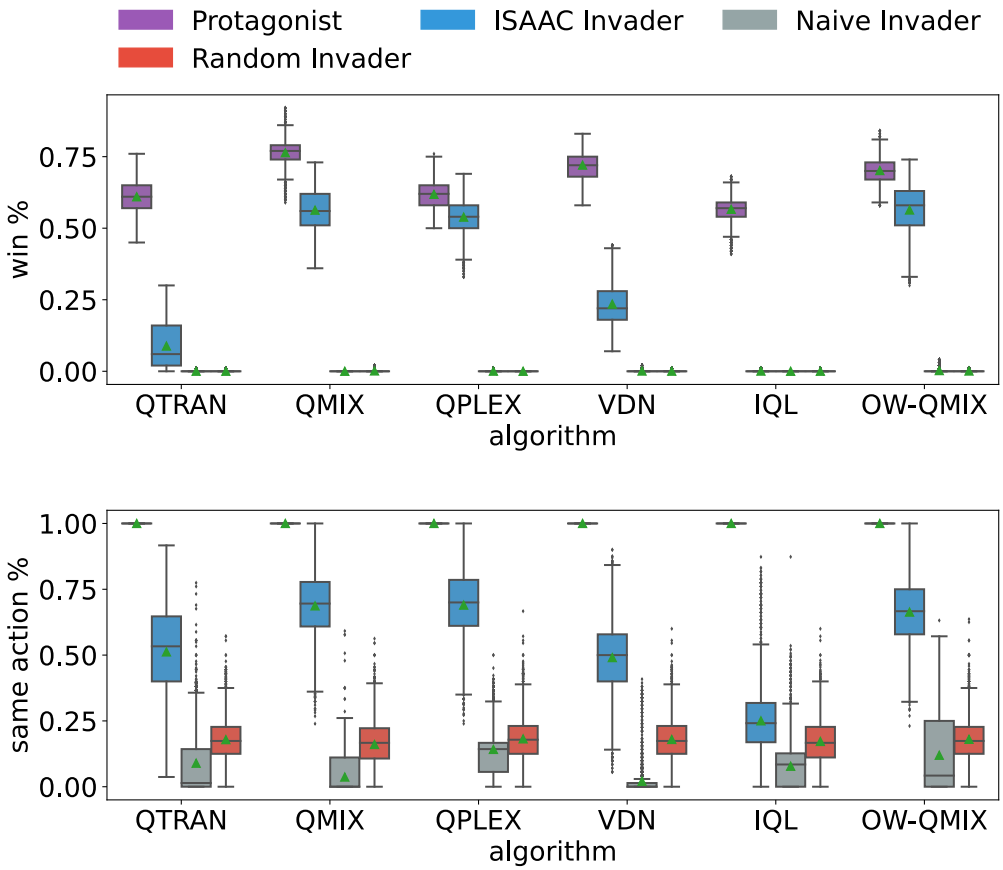


Figure A.7.: 5×1000 Evaluation episodes after training in 5m vs 6m of different algorithms.

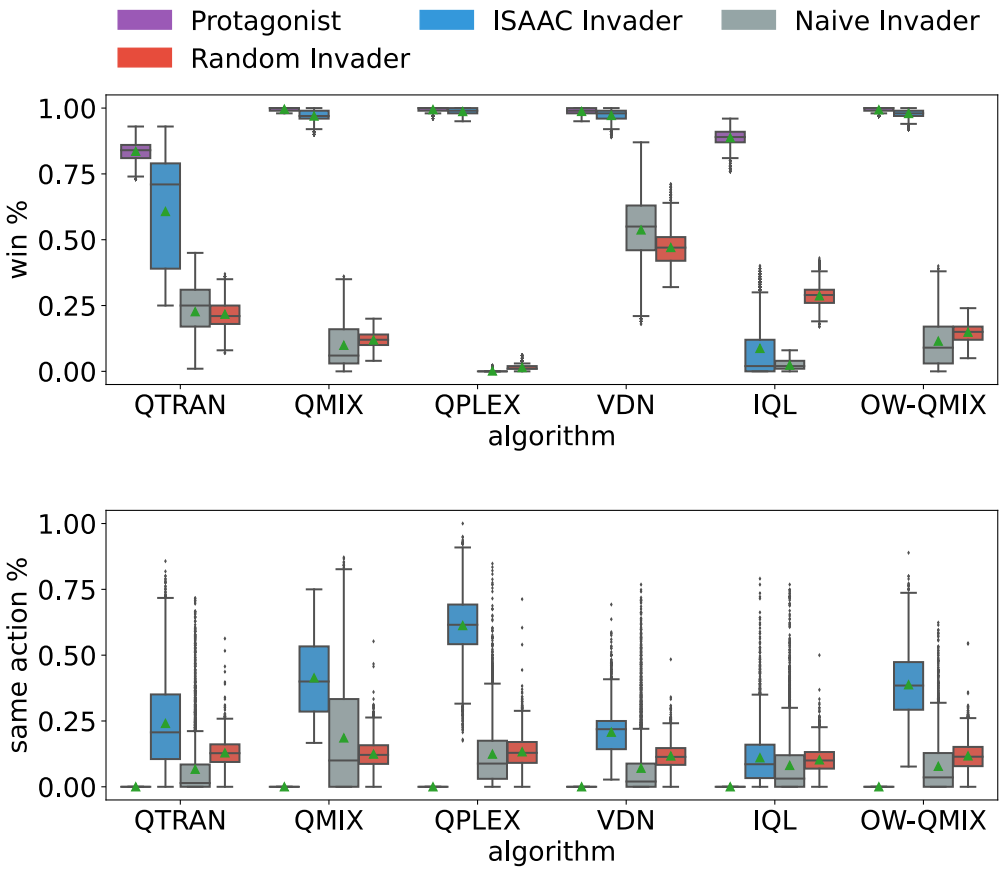


Figure A.8.: 5×1000 Evaluation episodes after training in MMM of different algorithms.

List of Figures

List of Tables

Bibliography

[ABC+20] Charles Audet, Jean Bigeon, Dominique Cartier, Sébastien Le Digabel, and Ludovic Salomon. Performance indicators in multiobjective optimization. *European Journal of Operational Research*, 292, 11 2020.

[ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.

[AFLZ22] Abolfazl Ahmadi, Mohammad Mahdi Forootan, Iman Larki, and Rahim Zahedi. Machine learning and deep learning in energy systems: A review. *Sustainability*, 14, 09 2022.

[AK19] Nahid Anwar and Susmita Kar. Review paper on various software testing techniques strategies. *Global Journal of Computer Science and Technology*, pages 43–49, 05 2019.

[And86] Charles W. Anderson. Learning and problem-solving with multilayer connectionist systems (adaptive, strategy learning, neural networks, reinforcement learning). 1986.

[And17] H. Anderson. Evading machine learning malware detection. 2017.

[ASM+11] Prince Agarwal, Manjari Sahai, Vaibhav Mishra, Monark Bag, and Vrijendra Singh. A review of multi-criteria decision making techniques for supplier evaluation and selection. *International Journal of Industrial Engineering Computations*, 2:801–810, 10 2011.

[Bar18] Samuel A. Barnett. Convergence problems with generative adversarial networks (gans). *CoRR*, abs/1806.11382, 2018.

[BEH+20] Yoram Bachrach, Richard Everett, Edward Hughes, Angeliki Lazaridou, Joel Z. Leibo, Marc Lanctot, Michael Johanson, Wojciech M. Czarnecki, and Thore Graepel. Negotiating team formation using deep reinforcement learning. *CoRR*, abs/2010.10380, 2020.

[Bel57] Richard Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957.

[BH22] Hojat Behrooz and Yeganeh M. Hayeri. Machine learning applications in surface transportation systems: A literature review. *Applied Sciences*, 12(18), 2022.

[BMR+20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020.

[BSGK22] Junyoung Byun, Kyujin Shim, Hyojun Go, and Changick Kim. Hidden conditional adversarial attacks. In *2022 IEEE International Conference on Image Processing (ICIP)*, pages 1306–1310, 2022.

[BZI13] Daniel S. Bernstein, Shlomo Zilberstein, and Neil Immerman. The complexity of decentralized control of markov decision processes. *CoRR*, abs/1301.3836, 2013.

[CDS19] Aidan Clark, Jeff Donahue, and Karen Simonyan. Efficient video generation on complex datasets. *CoRR*, abs/1907.06571, 2019.

[CEW11] Georgios Chalkiadakis, Edith Elkind, and Michael Wooldridge. Computational aspects of cooperative game theory. volume 5, 10 2011.

[Che21] Haiyang Chen. Challenges and corresponding solutions of generative adversarial networks (gans): A survey study. *Journal of Physics: Conference Series*, 1827(1):012066, mar 2021.

[CLX+19] Tong Chen, Jiqiang Liu, Yingxiao Xiang, Wenjia Niu, Endong Tong, and Zhen Han. Adversarial attack and defense in reinforcement learning-from ai security view. *Cybersecurity*, 2, 12 2019.

[CPLK20] Mingyu Cai, Hao Peng, Zhijun Li, and Zhen Kan. Learning-based probabilistic ltl motion planning with environment and motion uncertainties. *IEEE Transactions on Automatic Control*, PP, 07 2020.

[CvMBB14] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014.

[CW16] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. *CoRR*, abs/1608.04644, 2016.

[DMW+20] Ranjie Duan, Xingjun Ma, Yisen Wang, James Bailey, A. Kai Qin, and Yun Yang. Adversarial camouflage: Hiding physical-world attacks with natural styles. *CoRR*, abs/2003.08757, 2020.

[GAZ17] Amirata Ghorbani, Abubakar Abid, and James Zou. Interpretation of neural networks is fragile, 2017.

[GEB15] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015.

[GLZ+20] Ruize Gao, Feng Liu, Jingfeng Zhang, Bo Han, Tongliang Liu, Gang Niu, and Masashi Sugiyama. Maximum mean discrepancy is aware of adversarial attacks. *CoRR*, abs/2010.11415, 2020.

[Goo17] Ian J. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160, 2017.

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[GRBN09] Jinwei Gu, Ravi Ramamoorthi, Peter Belhumeur, and Shree Nayar. Removing image artifacts due to dirty camera lenses and thin occluders. *ACM Trans. Graph.*, 28(5):1–10, dec 2009.

[GSS14] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2014.

[Gun18] Nyoman Gunantara. A review of multi-objective optimization: Methods and its applications. *Cogent Engineering*, 5(1):1502242, 2018.

[GWH19] Jan Philip Göpfert, Heiko Wersing, and Barbara Hammer. Adversarial attacks hidden in plain sight. *CoRR*, abs/1902.09286, 2019.

[GY18] Abhishek Gupta and Zhaoyuan Yang. Adversarial reinforcement learning for observer design in autonomous systems under cyber attacks. *CoRR*, abs/1809.06784, 2018.

[HBD21] Keyang He, Bikramjit Banerjee, and Prashant Doshi. Cooperative-competitive reinforcement learning with history-dependent rewards. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '21, page 602–610, Richland, SC, 2021. International Foundation for Autonomous Agents and Multiagent Systems.

[HCC+21] Rongjie Huang, Chenye Cui, Feiyang Chen, Yi Ren, Jinglin Liu, Zhou Zhao, Baoxing Huai, and Zhefeng Wang. Singgan: Generative adversarial network for high-fidelity singing voice generation, 2021.

[HE16] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *CoRR*, abs/1606.03476, 2016.

[HEKK18] Sara Hooker, Dumitru Erhan, Pieter-Jan Kindermans, and Been Kim. Evaluating feature importance estimates. *CoRR*, abs/1806.10758, 2018.

[HPG+17] Sandy H. Huang, Nicolas Papernot, Ian J. Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *CoRR*, abs/1702.02284, 2017.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[HS15] Matthew J. Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *CoRR*, abs/1507.06527, 2015.

[HZ19] Yunhan Huang and Quanyan Zhu. Deceptive reinforcement learning under adversarial manipulations on cost signals. *CoRR*, abs/1906.10571, 2019.

[HZB+19] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. *CoRR*, abs/1907.07174, 2019.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[IRM+19] Mohsen Imani, Mohammad Saidur Rahman, Nate Mathews, Aneesh Yogesh Joshi, and Matthew Wright. Adv-dwf: Defending against deep-learning-based website fingerprinting attacks with adversarial traces. *CoRR*, abs/1902.06626, 2019.

[JAAA16] Abid Jamil, Muhammad Arif, Normi Abubakar, and Akhlaq Ahmad. Software testing techniques: A literature review. pages 177–182, 11 2016.

[JEP+21] John M. Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Zídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A A Kohl, Andy Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David A. Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with alphafold. *Nature*, 596:583 – 589, 2021.

[JSJ94] Tommi Jaakkola, Satinder P. Singh, and Michael I. Jordan. Reinforcement learning algorithm for partially observable markov decision problems. In *Proceedings of the 7th International Conference on Neural Information Processing Systems*, NIPS'94, page 345–352, Cambridge, MA, USA, 1994. MIT Press.

[KdW06] I. Y. Kim and Olivier L. de Weck. Adaptive weighted sum method for multiobjective optimization: a new method for pareto front generation. *Structural and Multidisciplinary Optimization*, 31:105–116, 2006.

[KGB16] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *CoRR*, abs/1611.01236, 2016.

[KGT+0] Sai Rahul Kaminwar, Jann Goschenhofer, Janek Thomas, Ingo Thon, and Bernd Bischl. Structured verification of machine learning models in industrial settings. *Big Data*, 0(0):null, 0. PMID: 34978896.

[Kis19] Michael Kissner. Hacking neural networks: A short introduction. *CoRR*, abs/1911.07658, 2019.

[KL51] Solomon Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:79–86, 1951.

[KLA18] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *CoRR*, abs/1812.04948, 2018.

[KNLH19] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey E. Hinton. Similarity of neural network representations revisited. *CoRR*, abs/1905.00414, 2019.

[KP19] Stepan Komkov and Aleksandr Petiushko. Advhat: Real-world adversarial attack on arcface face ID system. *CoRR*, abs/1908.08705, 2019.

[KS17] Jernej Kos and Dawn Song. Delving into adversarial attacks on deep policies, 2017.

[LHL+17] Yen-Chen Lin, Zhang-Wei Hong, Yuan-Hong Liao, Meng-Li Shih, Ming-Yu Liu, and Min Sun. Tactics of adversarial attack on deep reinforcement learning agents. *CoRR*, abs/1703.06748, 2017.

[Lin91] Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Trans. Inf. Theory*, 37:145–151, 1991.

[Lit94] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In William W. Cohen and Haym Hirsh, editors, *Machine Learning Proceedings 1994*, pages 157–163. Morgan Kaufmann, San Francisco (CA), 1994.

[LLM+22] Cameron Lischke, Tongtong Liu, Joe McCalmon, Md Asifur Rahman, Talal Halabi, and Sarra Alqahtani. Lstm-based anomalous behavior detection in multi-agent reinforcement learning. In *2022 IEEE International Conference on Cyber Security and Resilience (CSR)*, pages 16–21, 2022.

[LMK+17] Jan Leike, Miljan Martic, Victoria Krakovna, Pedro A. Ortega, Tom Everitt, Andrew Lefrancq, Laurent Orseau, and Shane Legg. AI safety gridworlds. *CoRR*, abs/1711.09883, 2017.

[LSC21] Ken Ming Lee, Sriram Ganapathi Subramanian, and Mark Crowley. Investigation of independent reinforcement learning algorithms in multi-agent environments. *CoRR*, abs/2111.01100, 2021.

[LZL+17] Joel Z. Leibo, Vinícius Flores Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas. *CoRR*, abs/1702.03037, 2017.

[McC95] R. Andrew McCallum. Instance-based utile distinctions for reinforcement learning with hidden state. In Armand Prieditis and Stuart J. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning (ICML 1995)*, pages 387–395, San Francisco, CA, USA, 1995. Morgan Kauffman.

[Mes18] Lars M. Mescheder. On the convergence properties of GAN training. *CoRR*, abs/1801.04406, 2018.

[MFF15] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. *CoRR*, abs/1511.04599, 2015.

[MKS+13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[MKS+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[MMS+17] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2017.

[MMS+19] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. A survey on bias and fairness in machine learning. *CoRR*, abs/1908.09635, 2019.

[MO14] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014.

[MRSW19] Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. MAVEN: multi-agent variational exploration. *CoRR*, abs/1910.07483, 2019.

[Nak05] Hirotaka Nakayama. Multi-objective optimization and its engineering applications. In Jürgen Branke, Kalyanmoy Deb, Kaisa Miettinen, and Ralph E. Steuer, editors, *Practical Approaches to Multi-Objective Optimization, 7.-12. November 2004*, volume 04461 of *Dagstuhl Seminar Proceedings*. IBFI, Schloss Dagstuhl, Germany, 2005.

[Net05] Goncalo Neto. From single-agent to multi-agent reinforcement learning: Foundational concepts and methods. 01 2005.

[NH19] Atsuhiro Noguchi and Tatsuya Harada. Image generation from small datasets via batch statistics adaptation. *CoRR*, abs/1904.01774, 2019.

[NIGM18] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *CoRR*, abs/1811.03378, 2018.

[NMSE20] Elior Nehemya, Yael Mathov, Asaf Shabtai, and Yuval Elovici. Taking over the stock market: Adversarial perturbations against algorithmic traders, 2020.

[Oli12] Frans A. Oliehoek. *Decentralized POMDPs*, pages 471–503. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[OSV11] Frans A. Oliehoek, Matthijs T. J. Spaan, and Nikos Vlassis. Optimal and approximate q-value functions for decentralized pomdps. *CoRR*, abs/1111.0062, 2011.

[PDSG17] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. *CoRR*, abs/1703.02702, 2017.

[PGS+20] Thomy Phan, Thomas Gabor, Andreas Sedlmeier, Fabian Ritz, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, Jan Wieghardt, Marc Zeller, and Claudia Linnhoff-Popien. Learning and testing resilience in cooperative multi-agent systems. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '20, page 1055–1063, Richland, SC, 2020. International Foundation for Autonomous Agents and Multiagent Systems.

[PP19] Roberta Pappadà and Francesco Pauli. Discrimination in machine learning algorithms. 01 2019.

[Pro07] Emily Pronin. Perception and misperception of bias in human judgment. *Trends in Cognitive Sciences*, 11:37–43, 2007.

[PTL+17] Anay Pattanaik, Zhenyi Tang, Shuijing Liu, Gautham Bommannan, and Girish Chowdhary. Robust deep reinforcement learning with adversarial attacks. *CoRR*, abs/1712.03632, 2017.

[QYZ+21] You Qiaoben, Chengyang Ying, Xinning Zhou, Hang Su, Jun Zhu, and Bo Zhang. Understanding adversarial attacks on observations in deep reinforcement learning. *CoRR*, abs/2106.15860, 2021.

[RBL+21] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. *CoRR*, abs/2112.10752, 2021.

[RDN+22] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents, 2022.

[RFPW20] Tabish Rashid, Gregory Farquhar, Bei Peng, and Shimon Whiteson. Weighted QMIX: expanding monotonic value function factorisation. *CoRR*, abs/2006.10800, 2020.

[RRD+20] Amin Rakhsha, Goran Radanovic, Rati Devidze, Xiaojin Zhu, and Adish Singla. Policy teaching via environment poisoning: Training-time adversarial attacks against reinforcement learning. *CoRR*, abs/2003.12909, 2020.

[RSdW+18] Tabish Rashid, Mikayel Samvelyan, Christian Schröder de Witt, Gregory Farquhar, Jakob N. Foerster, and Shimon Whiteson. QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. *CoRR*, abs/1803.11485, 2018.

[RSL18] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. *CoRR*, abs/1801.09344, 2018.

[Rub99] Reuven Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, 1(2):127–190, 1999.

[Rud16] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[SB98] R.S. Sutton and A.G. Barto. *Reinforcement learning: an introduction*. MIT Press, Cambridge, MA., 1998.

[SBBR16] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 1528–1540, New York, NY, USA, 2016. Association for Computing Machinery.

[SCHL21] Lucas Schott, Manon Césaire, Hatem Hajri, and Sylvain Lamprier. Improving robustness of deep reinforcement learning agents: Environment attacks based on critic networks. *CoRR*, abs/2104.03154, 2021.

[SGZ+16] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016.

[Sha48] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[SKK+19] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning, 2019.

[SLG+17] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinícius Flores Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning. *CoRR*, abs/1706.05296, 2017.

[SMRL59] Alvin Scodel, J. Sayer Minas, Philburn Ratoosh, and Milton E. Lipetz. Some descriptive aspects of two-person non-zero-sum games. *Journal of Conflict Resolution*, 3:114 – 119, 1959.

[SRdW+19] Mikayel Samvelyan, Tabish Rashid, Christian Schröder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philip H. S. Torr, Jakob N. Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. *CoRR*, abs/1902.04043, 2019.

[Sut88] Richard Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3:9–44, 08 1988.

[SVJ19] S. Reema Sree, S.B. Vyshnavi, and N. Jayapandian. Real-world application of machine learning and deep learning. In *2019 International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pages 1069–1073, 2019.

[SVS17] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *CoRR*, abs/1710.08864, 2017.

[SWW+19] Shawn Shan, Emily Willson, Bolun Wang, Bo Li, Haitao Zheng, and Ben Y. Zhao. Gotta catch 'em all: Using concealed trapdoors to detect adversarial attacks on neural networks. *CoRR*, abs/1904.08554, 2019.

[SZS+13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2013.

[SZZ18] Kun Shao, Yuanheng Zhu, and Dongbin Zhao. Starcraft micromanagement with reinforcement learning and curriculum transfer learning. *CoRR*, abs/1804.00810, 2018.

[Tan93] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Machine Learning Proceedings 1993*, pages 330–337. Morgan Kaufmann, San Francisco (CA), 1993.

[TTV18] Hoang Thanh-Tung, Truyen Tran, and Svetha Venkatesh. On catastrophic forgetting and mode collapse in generative adversarial networks. *CoRR*, abs/1807.04015, 2018.

[TWW+21] James Tu, Tsun-Hsuan Wang, Jingkang Wang, Sivabalan Manivasagam, Mengye Ren, and Raquel Urtasun. Adversarial attacks on multi-agent communication. *CoRR*, abs/2101.06560, 2021.

[UO12] Alberto Uriarte and Santiago Ontanon. Kiting in rts games using influence maps. pages 31–36, 01 2012.

[VAAMS22] Kevin Voogd, Jean Pierre Allamaa, Javier Alonso-Mora, and Tong Duy Son. Reinforcement learning from simulation to real world autonomous driving using digital twin, 2022.

[VBC+19] Oriol Vinyals, Igor Babuschkin, Wojciech Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John Agapiou, Max Jaderberg, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575, 11 2019.

[VH13] Mark Velasquez and Patrick Hester. An analysis of multi-criteria decision making methods. *International Journal of Operations Research*, 10:56–66, 05 2013.

[VRQI10] Sebastian Varges, Giuseppe Riccardi, Silvia Quarteroni, and Alexei Ivanov. The exploration/exploitation trade-off in reinforcement learning for dialogue management. pages 479 – 484, 01 2010.

[WD92] Christopher Watkins and Peter Dayan. Technical note: Q-learning. *Machine Learning*, 8:279–292, 05 1992.

[WFL+22] Kuan-Chieh Wang, Yan Fu, Ke Li, Ashish Khisti, Richard S. Zemel, and Alireza Makhzani. Variational model inversion attacks. *CoRR*, abs/2201.10787, 2022.

[WLS+22] Zhongshu Wang, Lingzhi Li, Zhen Shen, Li Shen, and Liefeng Bo. 4k-nerf: High fidelity neural radiance fields at ultra high resolutions, 2022.

[WPKD12] J.B. Williams, D. Popp, K.A. Kobak, and M.J. Detke. P-640 - the power of expectation bias. *European Psychiatry*, 27:1, 2012. Abstracts of the 20th European Congress of Psychiatry.

[WRL+20] Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. QPLEX: duplex dueling multi-agent q-learning. *CoRR*, abs/2008.01062, 2020.

[WZH17] Huikai Wu, Junge Zhang, and Kaiqi Huang. MSC: A dataset for macro-management in starcraft II. *CoRR*, abs/1710.03131, 2017.

[XEZJ22] Zikang Xiong, Joe Eappen, He Zhu, and Suresh Jagannathan. Defending observation attacks in deep reinforcement learning via detection and denoising, 2022.

[XI15] Shuo Xiong and Hiroyuki Iida. Attractiveness of real time strategy games. *2014 2nd International Conference on Systems and Informatics, ICSAI 2014*, 12:271–276, 01 2015.

[XLWW21] Wenju Xu, Chengjiang Long, Ruisheng Wang, and Guanghui Wang. DRB-GAN: A dynamic resblock generative adversarial network for artistic style transfer. *CoRR*, abs/2108.07379, 2021.

[XWF+18] Zheng Xu, Michael J. Wilber, Chen Fang, Aaron Hertzmann, and Hailin Jin. Beyond textures: Learning from multi-domain artistic images for arbitrary style transfer. *CoRR*, abs/1805.09987, 2018.

[YHZ+17] Xiaoyong Yuan, Pan He, Qile Zhu, Rajendra Rana Bhat, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *CoRR*, abs/1712.07107, 2017.

[YJvdS19] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. Time-series generative adversarial networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[YLD71] Yv Haimes Yv, Leon S. Lasdon, and Dang Da. On a bicriterion formation of the problems of integrated system identification and system optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, pages 296–297, 1971.

[ZCBH21] Huan Zhang, Hongge Chen, Duane S. Boning, and Cho-Jui Hsieh. Robust reinforcement learning on state observations with learned optimal adversary. *CoRR*, abs/2101.08452, 2021.

[ZDS17] Zhengli Zhao, Dheeru Dua, and Sameer Singh. Generating natural adversarial examples. *CoRR*, abs/1710.11342, 2017.

[ZHML19] Jie M. Zhang, Mark Harman, Lei Ma, and Yang Liu. Machine learning testing: Survey, landscapes and horizons. *CoRR*, abs/1906.10742, 2019.

[ZML16] Junbo Jake Zhao, Michaël Mathieu, and Yann LeCun. Energy-based generative adversarial network. *CoRR*, abs/1609.03126, 2016.

[ZWZ22] Jie Zhu, Fengge Wu, and Junsuo Zhao. An overview of the action space for deep reinforcement learning. In *2021 4th International Conference on Algorithms, Computing and Artificial Intelligence*, ACAI'21, New York, NY, USA, 2022. Association for Computing Machinery.