Curiosity-Driven and Victim-Aware Adversarial Policies

Chen Gong[*]
Institute of Automation, Chinese Academy of Sciences, China
gongchen2020@ia.ac.cn

Zhou Yang[†]
Singapore Management University
Singapore
zyang@smu.edu.sg

Yunpeng Bai
Institute of Automation, Chinese Academy of Sciences, China
baiyunpeng2020@ia.ac.cn

Jieke Shi
Singapore Management University
Singapore
jiekeshi@smu.edu.sg

Arunesh Sinha
Rutgers University, New Jersey, USA
arunesh.sinha@rutgers.edu

Bowen Xu
Singapore Management University
Singapore
bowenxu@smu.edu.sg

David Lo
Singapore Management University
Singapore
davidlosmu.edu.sg

Xinwen Hou
Institute of Automation, Chinese Academy of Sciences, Beijing, China
xinwen.hou@ia.ac.cn

Guoliang Fan
Institute of Automation, Chinese Academy of Sciences, Beijing, China
guoliang.fan@ia.ac.cn

ABSTRACT

Recent years have witnessed great potential in applying Deep Reinforcement Learning (DRL) in various challenging applications, such as autonomous driving, nuclear fusion control, complex game playing, etc. However, recently researchers have revealed that deep reinforcement learning models are vulnerable to adversarial attacks: malicious attackers can train *adversarial policies* to tamper with the observations of a well-trained victim agent, the latter of which fails dramatically when faced with such an attack. Understanding and improving the adversarial robustness of deep reinforcement learning is of great importance in enhancing the quality and reliability of a wide range of DRL-enabled systems.

In this paper, we develop *curiosity-driven* and *victim-aware* adversarial policy training, a novel method that can more effectively exploit the defects of victim agents. To be victim-aware, we build a surrogate network that can approximate the state-value function of a black-box victim to collect the victim's information. Then we propose a curiosity-driven approach, which encourages an adversarial policy to utilize the information from the hidden layer of the surrogate network to exploit the vulnerability of victims efficiently. Extensive experiments demonstrate that our proposed method *outperforms or achieves a similar level of performance as the current state-of-the-art* across multiple environments. We perform an ablation study to emphasize the benefits of utilizing the approximated victim information. Further analysis suggests that our method is harder to defend against a commonly used defensive strategy, which calls attention to more effective protection on the systems using DRL.

CCS CONCEPTS

• **Computing methodologies → Planning and scheduling**; • **Security and privacy → Domain-specific security and privacy architectures**.

KEYWORDS

Adversarial Attack, Reinforcement Learning, Curiosity Mechanism

ACM Reference Format:
Chen Gong, Zhou Yang, Yunpeng Bai, Jieke Shi, Arunesh Sinha, Bowen Xu, David Lo, Xinwen Hou, and Guoliang Fan. 2022. Curiosity-Driven and Victim-Aware Adversarial Policies. In *Annual Computer Security Applications Conference (ACSAC '22), December 5–9, 2022, Austin, TX, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3564625.3564636

1 INTRODUCTION

Deep Reinforcement Learning (DRL), which utilizes Deep Neural Networks (DNN) to learn optimal policies, has demonstrated human-level performance in tackling many challenging tasks and has been adopted in critical domains, including complex game playing [32, 41, 53], autonomous driving [39, 60], security [12, 21, 22, 50], robotics control [19, 23, 33], and even nuclear fusion [11].

Although the success of DRL is evident in many domains, DRL, just like many other DNN models [31, 58], has recently been shown to be vulnerable to adversarial attacks [25, 51, 57]. For example, by adding carefully-chosen noises to the background image in a game, an attacker can drive a well-trained agent to choose sub-optimal actions with high confidence [25]. However, attacking by adding adversarial perturbations to victim inputs is considered unrealistic in practice as it requires an attacker to hack into the system (e.g., game engines). A more practical method is to attack via training *adversarial policies*. It assumes that a malicious user can control one player in a two-player game, and this player fools the deployed

[*] This paper was done when Chen Gong was a visiting student at SMU.
[†] The corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACSAC '22, December 5–9, 2022, Austin, TX, USA
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9759-9/22/12…$15.00
https://doi.org/10.1145/3564625.3564636

victim agent by taking legal actions allowed in the environment [15]. Such adversarial policies follow a purely black-box manner where the victim agent's parameters are unavailable. Prior studies [15, 20] assume that the victim agent takes a fixed policy, which is reasonable in the scenario of deep learning model testings [63]. For example, developers are not likely to allow an autonomous driving car to update its policy when it is deployed on the road, as untested changes to a safety-critical system may cause serious accidents. Having such an assumption, researchers can convert the process of training adversarial policies in a multi-agent environment into a single-agent DRL problem and show that resultant adversarial agents can achieve decent winning rates [15, 20].

The success of adversarial policies can be attributed to the insight that an attacker can manipulate the victim agent's observation by taking uncommon actions to lead the game into unfamiliar states, and as a result making the victim exhibit undesired sub-optimal behaviours. This suggests that an attacker's capability in exploring such states may affect the effectiveness of the attack, motivating us to encourage the training of adversarial policies that can better explore the environment. In the conventional DRL research, researchers have proposed the *curiosity mechanism* to encourage exploration [8]: adding larger *intrinsic rewards* to the states that agents are unfamiliar with. The combined (intrinsic and extrinsic) reward provides agents with more incentives to explore unfamiliar states during the training stage. It motivates us to present a novel *victim-aware* and *curiosity-driven* approach to training adversarial policies that can efficiently explore the vulnerability of victims.

Similar to practices adopted by Gleave et al. [15] and Guo et al. [20], we fix the victim's policy and convert the training problem of finding the adversarial policy in a two-player game into a single-agent learning problem. Then, we incorporate Random Network Distillation (RND), a popular framework to encourage exploration [8] by assigning specially designed intrinsic rewards for different states to encourage exploration. The RND approach introduces two randomly initialized networks: one with fixed parameters is called the *target* network, and the other one is the *predictor* network which learns to predict outputs of the target network given states as input. Intuitively, if a state is barely visited, the difference between the two networks' outputs is expected to be large. Those "unfamiliar" states are given larger intrinsic rewards to encourage the adversarial policy to explore such states.

Although standard RND can encourage an agent to explore the environment, it is not designed for exploring the vulnerability of victims under adversarial settings. A victim's state-value function, which describes the expected rewards an agent will receive if it enters a state, contains rich information pertaining to the vulnerability of the victim, e.g., the importance of states to the victim. An attacker should explore unfamiliar states that are useful in lowering the victim's expected rewards. As we assume a black-box attack scenario where the victim's value network is unknown, we first build a surrogate network to approximate the victim state-value function (explained in Section 3.1). Then, we propose to feed the outputs of a hidden layer in the victim value network to the target and predictor networks in RND rather than directly input the raw observation or some observation features obtained from a static feature extractor without information about the victim. The advantage of this *victim-aware* RND is that the hidden layer contains information about the victim that can be utilized to guide the exploration of the victim's vulnerabilities.

We perform detailed experiments to evaluate our curiosity-driven and victim-aware adversarial policies. We choose the approach proposed by Guo et al. [20], which is state-of-the-art, as the baseline. The experiment results empirically show the advantages of using the victim-aware curiosity mechanism. The evaluation is conducted in both zero-sum (MuJoCo [5]) and non-zero-sum games (Starcraft II [43]). The goal of the attacker is to win by defeating the opponent (i.e., the victim agent). So we use the winning rate of the attacker as the main metrics to evaluate the performance of an adversarial policy training method. As demonstrated in Section 5.1, adversarial policies trained with the proposed method can *outperform* (or at least maintain a similar level of performance as) the baseline method across all the games investigated, and the average winning rates of adversarial policy trained using our method is 7.3% higher than that of the baseline [20]. Although Gleave et al. [15] point out the difficulty of training adversarial policies in games where the dimension of the agent's observation is low, our proposed method is the *first* to train adversarial policies that can work in low-dimensional environments like `Run-To-Goal-Ant`, where our adversarial policy's average winning rate improves by 24% than the state-of-the-art method [20].

An ablation study (Section 6.1) shows that compared with directly using raw states as input to RND, using outputs of the hidden layer in the surrogate network that approximates the victim state-value function leads to the more powerful adversarial policies, which improves the attacker's average winning rate by 5.8%. We also evaluate a defensive strategy [15, 20] called the observation masking strategy, i.e., the victim cannot see the adversary's position and behaviours. The results show that our proposed method is harder to defend by the observation masking strategy.

We organize the rest of this paper as follows. Section 2 formalizes the task of training adversarial policies in a two-player game. Section 3 elaborates on how to train the curiosity-driven and victim-aware adversarial policy. We introduce the experiment settings in Section 4, after which we present extensive experimental results for evaluating our method in Section 5 and conduct additional experiments to further analyze the contribution of the victim-aware mechanism in Section 6. Section 7 presents works that are related to this study. Finally, we conclude our paper in Section 8. The code and documentation, along with the obtained models, are made open-source.[1]

2 TASK FORMULATION

This section discusses deep reinforcement learning and formalizes the training of adversarial policies in a two-player game.

2.1 Deep Reinforcement Learning

Deep reinforcement learning trains an agent to solve a sequence decision problem that can be formalized as a Markov Decision Process (MDP). An MDP can be described using a 4-tuple: $\langle S, A, P, R\rangle$. S is a set of states, and A is a set of actions. P: $S \times A \rightarrow S$ is a transition function; $P(s'|s,a)$ refers to the probability that the environment transits from s to s' when an agent takes action $a \in A$ in s.

[1] https://github.com/soarsmu/Curiosity_in_Adversarial_Policy

$R: S \times A \to \mathbb{R}$ specifies a reward function, and $r = R(s, a)$ indicates that the agent receives a reward r after taking action a in state s.

An agent adopts a *policy* $\pi(\cdot)$ specifying what actions to take under a certain state. Specifically, given a state s_t at the time step t, the output of the policy $\pi(\cdot|s_t)$ is a probability distribution over all actions. The agent samples an action a_t according to the distribution, i.e., $a_t \sim \pi(\cdot|s_t)$. Then, the agent will receive a reward $r_t = R_t(s_t, a_t)$ from the environment. Meanwhile, the environment transits to a new state $s_{t+1} \sim P(\cdot|s_t, a_t)$. The agent starts at an *initial* state s_0 and interacts with the environment until it encounters a *termination* state s_T. The interaction history is called a *trajectory*:

$$\tau : (\langle s_0, a_0, r_0 \rangle, \langle s_1, a_1, r_1 \rangle, \cdots, s_T) \tag{1}$$

We formulate the *cumulative discounted return*, the sum of discounted rewards over a trajectory, as $R(\tau) = \sum_{i=0}^{T} \gamma^i R_t(s_i, a_i)$. $\gamma \in (0, 1)$ is the discount factor, denoting to what extent an agent cares about the rewards in the future. DRL aims to train an agent to find the optimal policy π^* that maximizes the expectation of the cumulative discounted return over all trajectories. Assuming τ^π is a trajectory generated using a policy π, the goal of DRL can be formalized as follows:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau\sim\pi}\left[R(\tau^\pi)\right] \tag{2}$$

Sutton et al. [44] prove that finding the optimal policy is equivalent to maximize the *state-value* function $V_\pi(s)$, which is defined as

$$V_\pi(s) = \mathbb{E}_{\tau\sim\pi}\left[R(\tau^\pi)|s_0 = s\right] \tag{3}$$

Given a state s, $V_\pi(s)$ indicates the expected cumulative discounted rewards over the trajectories that start from s and are generated by the policy π. Intuitively speaking, it approximates the value of a state, and the agent tends to enter the states with higher values.

2.2 Adversarial Policies in Two-Player Games

Unlike prior works that fool victim agents into choosing sub-optimal actions by perturbing the policy network input [25], we train *adversarial policies* to attack victims in a two-player competitive game. We formalize the problem of finding adversarial policies as follows.

In a two-player game where a well-trained victim agent v adopts a fixed policy, an attacker agent α defeats the victim agent v by taking an adversarial policy $\pi^\alpha(a|s)$, which describes the probability of taking the action a under the state s. In zero-sum games like MuJoCo [5], the attacker can maximize its winning rates by learning policies to maximize its own expected rewards. To enhance generalizability, this paper also investigates a non-zero-sum game (StarCraft [43]). However, due to the nature of non-zero-sum games, merely maximizing the rewards of attackers does not necessarily result in a corresponding reduction of winning rates on the victim side. Therefore, it is critical to train adversarial policies that can maximize the attacker's expected rewards $V_\pi^\alpha(s)$ while exerting a negative influence on the victim agent, i.e., minimizing the victim's expected rewards $V_\pi^v(s)$.

As this paper makes a realistic black-box assumption, an attack has no knowledge about the victim policy, making the joint policy unavailable. However, following past practice [20], we keep the victim's policy fixed during the training of adversarial policies. It is common sense that in a two-player Markov game, where the one agent (i.e., the victim) takes a fixed policy, the state transition function only depends on the policy of the other agent (i.e., the attacker). In such a setting, following the results in [20], we can define the victim state-value function as explicitly dependent on the state and attacker's policy. In this way, we can obviate the requirement for explicit knowledge of the victim's policy. After treating the victim with a fixed policy as part of the environment, we convert the two-player game into a single-agent DRL problem. Formally, the objective function in this DRL problem can be formalized as:

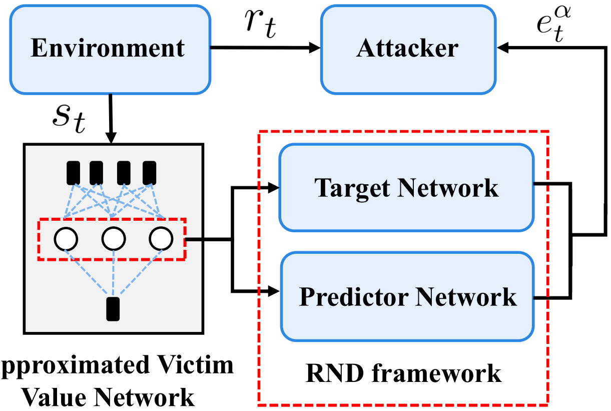


Figure 1: The architecture of curiosity-driven and victim-aware adversarial policies. Given the state s_t at time step t, the approximated victim value network provides an approximate value. One hidden layer output of the approximated value network is fed into the RND framework consisting of a target and predictor network to calculate the intrinsic reward e_t^α. Attacker takes the external reward r_t and e_t^α to decide actions and update the adversarial policy.

$$\arg\max_{\theta}\left(V^\alpha_{\pi^\alpha_\theta}(s) - V^v_{\pi^\alpha_\theta}(s)\right) \tag{4}$$

where $V^\alpha_{\pi^\alpha_\theta}(s)$ and $V^v_{\pi^\alpha_\theta}(s)$ represent the state-value function belonging to the attacker and victim respectively, and π^α_θ refers to the adversarial policy network with parameters θ. The detailed formulation of the state-value function is elaborated in Eqn (5). This objective function addresses the concerns of lack of knowledge about the victim agent and turns the training of the adversarial policy into a tractable problem. According to a recent empirical study [1], the proximal policy optimization (PPO) [40] algorithm is one of the most widely used algorithms in the DRL research, so we use PPO to train the adversarial policies in this paper.

3 CURIOSITY-DRIVEN AND VICTIM-AWARE ADVERSARIAL POLICIES

This section describes our curiosity-driven and victim-aware method to train adversarial policies that can tackle the problem described in Section 2.2. The curiosity-driven mechanism can encourage the attacker to explore various states to uncover the vulnerabilities of the victim. The victim-aware design allows the attacker to leverage the victim information hidden in the victim's state-value function.

Figure 1 depicts a high-level overview of our approach. In the figure, we first build a surrogate network to approximate the victim state-value function (Section 3.1) and then let the Random Network

Distillation (RND) framework (including a target and a predictor network) utilize the surrogate network's hidden layer output to produce the intrinsic reward to encourage exploration (Section 3.2). In Section 3.3, we explain the new objective function that considers intrinsic rewards and the training of adversarial policies.

3.1 Approximating Victim Information

As mentioned in Section 1, this paper assumes that an attacker cannot access any internal parameters or training details of a victim. Some works [6, 65] therefore proposed to build an imitation learning model that leverages the previous actions and states to predict future victim actions. However, such approaches only assist attackers by exposing the action information of the victim agent, which is not applicable in our setting as we need the victim's state-value function to solve Eqn (4). Section 2.2 points out that the output of the victim state-value function only depends on the state and the attacker policy, which brings about the possibility of approximating the victim state-value function. We make an additional assumption that the external rewards are known, which does not violate our black-box attack assumption and is also practical considering that the environment information is usually made publicly available. Then, we build a surrogate network to approximate the state-value function of the victim as follows.

At each time step t, given the current state s_t and the external reward r^{ν} to the victim, we can approximate the victim state-value function $V^{\nu}(\cdot)$ by Temporal Difference (TD) learning [46]. More specifically, we formally define the victim's state-value function:

$$V^{\nu}_{\pi^{\alpha}}(s_t) = \mathbb{E}_{\substack{a^{\alpha}\sim\pi^{\alpha}\\ a^{\nu}\sim\pi^{\nu}}}\left[R^{\nu}(s_t, a) + \gamma\mathbb{E}_{s_{t+1}\sim P(\cdot|s_t,a)}[V^{\nu}_{\pi^{\alpha}}(s_{t+1})]\right] \quad (5)$$

where π^{α} and π^{ν} represents the adversarial policy of the attacker and the victim's fixed policy, respectively. a^{α} and a^{ν} are the actions sampled from the adversarial policy and victim's policy, denoted by $a^{\alpha} \sim \pi^{\alpha}$ and $a^{\nu} \sim \pi^{\nu}$. $R^{\nu}(s_t, a)$ is a reward function to get the external reward r^{ν} to the victim agent in state s_t for the joint action $a = (a^{\alpha}, a^{\nu})$. $P(s_{t+1}|s_t, a)$ denotes the probability of transiting to a state s_{t+1} given the current state s_t and the joint action $(a^{\alpha}_t, a^{\nu}_t)$. γ is the discount factor. Using the TD-learning paradigm, we start by initializing an arbitrary neural network parameterized with θ^{ν} to model $V^{\nu}_{\pi^{\alpha}}(s)$, and then update the state-value function network for next state repeatedly by minimizing the TD-error, namely:

$$\underset{\theta^{\nu}}{\operatorname{argmin}} \left\|V^{\nu}_{\pi^{\alpha}}(s_t) - \left(R^{\nu}(s_t, a_t) + \gamma\mathbb{E}_{s_{t+1}\sim P}[V^{\nu}_{\pi^{\alpha}}(s_{t+1})]\right)\right\|^2 \quad (6)$$

where the joint action $a_t = (a^{\alpha}_t, a^{\nu}_t)$ is sampled from the attacker's and victim's policy. By doing so, we can approximate the victim state-value function during the training of adversarial policy.

3.2 Curiosity-Driven Victim-Aware Exploration

A successful attack can be attributed to the insight that the attacker has triggered an uncommon state that the victim is unfamiliar with (we elaborated on this viewpoint in Section 1). Such insight motivates us to encourage an adversarial policy to explore environments and encounter more states. We incorporate RND [36], a popular framework in exploration research, into adversarial training.

We include two randomly initialized neural networks: a fixed *target* network g and a *predictor* network $\hat{g}$ that tries to learn the output of g. As the victim state-value function captures the importance of a state s to the victim, it can be valuable to the attacker. Thus, we use the output of a hidden layer (specifically, the penultimate layer) from the approximated victim value network, which is represented as $\phi(s)$, and feed it to both the target and predictor networks. We view the state-value function network as feature extractor layers followed by layers to process the output of the feature extractor; as a consequence, the penultimate hidden layer contains the compact and relevant information needed for estimating the state value.

Let the target network g and predictor network $\hat{g}$ be parameterized by θ_g and $\theta_{\hat{g}}$, respectively. The goal is to train the predictor network to minimize the expected mean square error between the two networks. As illustrated in Figure 1, the expected mean square error is also utilized as the intrinsic reward e^{α} to drive the attacker's exploration, defined as:

$$e^{\alpha} = \left\|\hat{g}_{\theta_{\hat{g}}}(\phi(s)) - g_{\theta_g}(\phi(s))\right\|^2. \quad (7)$$

The intrinsic reward for the "novel" inputs should be higher than inputs that have been seen and used to train the predictor network [8], which therefore drives the attacker to perform actions leading to those novel states that the victim is not familiar with.

3.3 Adversarial Policies Training

Following the technique in [20] to train adversarial policy using the PPO algorithm [40], we combine approximated victim state-value function with the intrinsic rewards and then reformulate the objective in Eq. (4) as follows:

$$\begin{aligned}
&\arg\max_{\theta}\mathbb{E}_{(a^{\alpha}_t,s_t)\sim\pi^{\alpha}_{\text{old}}}\left[\min\left(\text{clip}(\rho_t, 1-\epsilon, 1+\epsilon)A^{\alpha}_t, \rho_t A^{\alpha}_t\right)\right.\\
&\left.-\min\left(\text{clip}(\rho_t, 1-\epsilon, 1+\epsilon)A^{\nu}_t, \rho_t A^{\nu}_t\right)\right],\\
&\rho_t = \frac{\pi^{\alpha}_{\theta}(a^{\alpha}_t|s_t)}{\pi^{\alpha}_{\text{old}}(a^{\alpha}_t|s_t)}, A^{\nu}_t = A^{\nu}_{\pi^{\alpha}_{\text{old}}}(a^{\alpha}_t, s_t),\\
&A^{\alpha}_t = A^{\alpha}_{\pi^{\alpha}_{\text{old}}}(a^{\alpha}_t, s_t) + \lambda A^{\alpha,\text{ins}}_{\pi^{\alpha}_{\text{old}}}(a^{\alpha}_t, s_t)
\end{aligned} \quad (8)$$

In the above formulas, π^{α}_{θ} denotes the adversarial policy network parameterized by θ. $\pi^{\alpha}_{\text{old}}$ and π^{α}_{θ} refer to the old and new adversarial policy, respectively. A^{ν}_t and A^{α}_t represent the *advantage functions* (a part of PPO) of the victim and the attacker. We have $A^i_t = r^i_t + \gamma V^i(s_{t+1}) - V^i(s_t), i \in \{\alpha, \nu\}$. Note that our victim-aware curiosity mechanism introduces an additional advantage function: $A^{\alpha,\text{ins}}$, which is derived from the intrinsic rewards. λ is a hyper-parameter representing the degree of exploration. ϵ constrains the interval of the probability ratio in the PPO algorithm. The last row of the Eqn (8) follows the advantage decomposition in Burda et al. [8], which defines the advantage as the sum of the advantages of intrinsic and extrinsic rewards. It should be noted that the intrinsic rewards do not introduce any negative impact on our training process, as the PPO algorithm guarantees a small policy update (within a trust region) at each timestep.

We summarize the proposed adversarial policies training method in Algorithm 1. There are six neural networks involved in our algorithm: (1) the adversarial policy π^{α}_{θ}; (2) the state-value function $V^{\alpha}_{\pi^{\alpha}}$ of the attacker α; (3) the approximated state-value function $V^{\nu}_{\pi^{\alpha}}$ of the victim ν; (4) the attacker's state-value function $V^{\alpha}_{\pi^{\alpha},\text{ins}}$ after considering intrinsic rewards; (5) the target network g_{θ_g} in RND, which is randomly initialized and fixed; and (6) the predictor network $\hat{g}_{\theta_{\hat{g}}}$

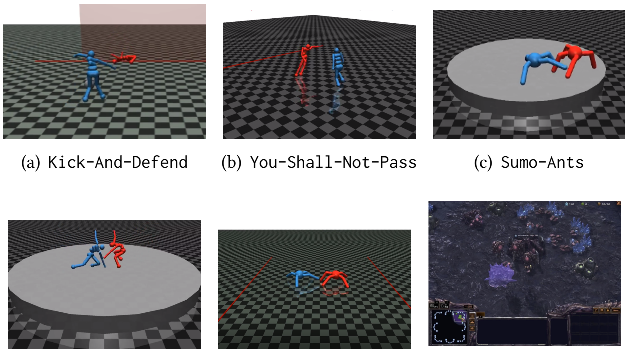
Algorithm 1: Curiosity-Driven and Victim-Aware Adversarial Policy

```
1  Input: $\pi_\theta^\alpha$: the adversarial policy parameterized by $\theta$; the
       state-value functions $V_{\pi^\alpha}^\alpha$, $V_{\pi^\alpha}^\nu$ and $V_{\pi^\alpha,\text{ins}}^\alpha$, with
       parameters $v_\alpha, v_\nu, v_{\text{ins}}$, respectively; $\phi$: the penultimate
       hidden layer of the network to approximate the victim's
       state-value function; $g_{\theta_g}$: the target network; $\hat{g}_{\theta_{\hat{g}}}$: the
       predictor network; $\gamma$: the discount factor; $T$: the total
       number of iteration.
2  Initialization: initialize $\theta_0, v_\nu^{(0)}, v_\alpha^{(0)}, v_{\text{ins}}^{(0)}, \theta_g, \theta_{\hat{g}}^{(0)}, t = 0$.
3  for $t = 1, 2, 3, \cdots, T$ do
4      # 1. Let the adversarial policy $\pi_{\theta_t}^\alpha$ play with the victim,
         and collect a set of trajectories $\mathcal{D}^{(t)} = \{\tau_i\}$, where
         $i = 1, 2, \cdots, |\mathcal{D}^{(t)}|$.
5      # 2. For each trajectory $\tau_i$, compute the intrinsic rewards
         $e_{i_k}^\alpha$ at each time step $k$ using Eqn (7).
6      # 3. For each trajectory $\tau_i$, compute the advantage at
         each time step $k$:
7      $A_{i_k}^\alpha = r_{i_k}^\alpha + \gamma V_{\pi^\alpha}^{\alpha(t)}\left(s_{i_{k+1}}^\alpha\right) - V_{\pi^\alpha}^{\alpha(t)}\left(s_{i_k}^\alpha\right)$,
8      $A_{i_k}^\nu = r_{i_k}^\nu + \gamma V_{\pi^\alpha}^{\nu(t)}\left(s_{i_{k+1}}^\alpha\right) - V_{\pi^\alpha}^{\nu(t)}\left(s_{i_k}^\alpha\right)$,
9      $A_{i_k}^{\alpha,\text{ins}} = e_{i_k}^\alpha + \gamma V_{\pi^\alpha,\text{ins}}^{\alpha(t)}\left(s_{i_{k+1}}^\alpha\right) - V_{\pi^\alpha,\text{ins}}^{\alpha(t)}\left(s_{i_k}^\alpha\right)$.
10     # 4. Then we have $A_{i_{0:|\tau_i|}}^\alpha$, $A_{i_{0:|\tau_i|}}^\nu$ and $A_{i_{0:|\tau_i|}}^{\alpha,\text{ins}}$
         $(i = 1, \cdots, |\mathcal{D}^{(t)}|)$ required in Eqn. (8) and then can
         obtain a new policy by maximizing the objective
         function in Eqn. (8).
11     # 5. Update $v_\nu^{(t)}, v_\alpha^{(t)}, v_{\text{ins}}^{(t)}$ by minimizing the TD error:
         $\frac{1}{|\mathcal{D}(t)|\times|\tau_i|}\sum_{i=0}^{|\mathcal{D}(t)|}\sum_{k=0}^{|\tau_i|}\left(V(s_{i_k}) - (r_{i_k} + \gamma V(s_{i_{k+1}}))\right)^2$.
12     # 6. Update $\theta_{\hat{g}}^{(t)}$ by minimizing the loss:
         $\frac{1}{|\mathcal{D}(t)|\times|\tau_i|}\sum_{i=0}^{|\mathcal{D}(t)|}\sum_{k=0}^{|\tau_i|}\left\|g(\phi(s_{i_k})) - \hat{g}(\phi(s_{i_k}))\right\|^2$.
13 end
14 Output: $\pi_\theta^\alpha$: the well trained adversarial policy network.
```

in RND. As shown in Line 4, we first let the adversarial policy play with the victim agent and collect a set of trajectories, denoted by $\mathcal{D}^{(t)} = \{\tau_i\}$, where $i = 1, 2, \cdots, |\mathcal{D}^{(t)}|$. A trajectory is a sequence of game states, actions, and rewards to players, represented as:

$$\tau_i = \left(s_{i_0}^\alpha, a_{i_0}^\alpha, \left\langle r_{i_0}^\alpha, r_{i_0}^\nu \right\rangle, \cdots, s_{|\tau_i|}^\alpha, a_{|\tau_i|}^\alpha, \left\langle r_{|\tau_i|}^\alpha, r_{|\tau_i|}^\nu \right\rangle\right). \tag{9}$$

At Line 5, for each trajectory τ_i, we get the intrinsic rewards by computing the expected mean square error between the two target and predictor networks. Then (Line 6 to 9), we are able to compute the advantage functions required in Eqn. (8). After having the necessary advantage functions to compute the objective of the adversarial policies, we can update π_θ^α to maximize the objective (Line 10). We update the parameters of state-value functions $V_{\pi^\alpha}^\alpha$, $V_{\pi^\alpha,\text{ins}}^\alpha$, $V_{\pi^\alpha}^\nu$, and predictor network $\hat{g}_{\theta_{\hat{g}}}$ accordingly (Line 11-12). After the above iterative process is finished, the algorithm returns π_θ^α as the adversarial policy network.



(a) Kick-And-Defend (b) You-Shall-Not-Pass (c) Sumo-Ants

(d) Sumo-Humans (e) Run-To-Goal-Ants (f) StarCraft II

Figure 2: The snapshots of six games investigated in our experiments. The first five games are from MuJoCo and the last game is StarCraft II.

4 EXPERIMENT SETUP

This section discusses experiment settings to evaluate our proposed adversarial policy training method, including the evaluation environments and the evaluation metrics.

4.1 Experiment games

Our experiments are conducted in six games: five zero-sum games from MuJoCo [5] (Kick-And-Defend, Sumo-Ants, Sumo-Humans, You-Shall-Not-Pass, and Run-To-Up-Ants) and a non-zero-sum game from Starcraft II [43]. All selected games are commonly adopted to evaluate DRL algorithms in two-player games. Figure 2 depicts the snapshots of six games. The MuJoCo contains a series of robotic control games, which are used as the representatives of zero-sum games. The agents in the selected MuJuCo games can be divided into two types: (1) Humanoid, a bipedal robot with 17 joints; and (2) Ant, the quadrupedal robot with 8 joints. StarCraft II is a real-time strategy game, which stands for non-zero-sum games. We briefly describe the investigated two-player games as follows:

- Kick-And-Defend: The kicker (the blue robot in Figure 2 (a)) aims to shoot a ball into the target region. The defender (the red robot) prevents the kicker from shooting successfully. Adopting the same setting as the baseline [20], we select the kicker as the victim agent.
- You-Shall-Not-Pass: As illustrated in Figure 2 (b), the runner (the blue robot) intends to run and cross the finish line (the red line). The blocker (the red robot) prevents the runner from crossing the finish line. Following the setting in [15, 20], we select the runner as the victim agent.
- Sumo-Ants and Sumo-Humans: Two humanoid and ant robots in a circular arena space (see in Figure 2 (c)/(d)) aim to knock down or push their opponent out of the arena.
- Run-To-Goal-Ants: Two ant robots are placed face to face in a flat space. The agent that reaches the finish line (the red line in Figure 2 (e)) first is the winner.
- StarCraft II: As presented in Figure 2 (f), the two agents manipulate the soldiers to take actions to defeat their opponent within a limited time. In the games shown in Figure 2 (c) to (f),

the goal of each player is the same. We randomly select one player as the attacker, and the other as the victim.

According to the size of observation of agents, the MuJoCo environments can be categorized into low-dimensional and high-dimensional ones. The observations in three humanoid environments are vectors of size 384, 395, and 380, respectively. The observations of the two ant environments are of lower dimensionality: `Sumo-Ants` has 137-dimensional observations and `Run-To-Go-Ants` has 122-dimensional observations. Following the setup in [15], we explicitly define the former three environments as high-dimensional and the latter two ant environments as low-dimensional. Besides, in StarCraft II, the size of observation for the agent is 857. We put more details of the selected environments in Supplementary A.1.

4.2 Evaluation Metrics

We define two the metrics to evaluate the performance of adversarial policies: the *winning rate* and the *non-loss rate*. The *winning rate* is formulated as: $\frac{\text{the number of winning rounds}}{\text{the number of total rounds}} \times 100\%$. A higher winning rate indicates that an adversarial agent is more powerful in exploring the weakness of the victim agent. Similarly, we define the *non-loss rate* as: $\frac{\text{the number of non-loss rounds}}{\text{the number of total rounds}} \times 100\%$. Following the setting in the baseline [20], we let each game run for 100 rounds and compute the two evaluation metrics.

5 EXPERIMENTAL RESULTS AND ANALYSIS

This section evaluates our adversarial attack method in the MuJoCo and StarCraft II environments. We compare the proposed method with the state-of-the-art baseline [20] from multiple perspectives.

5.1 Winning Rates of Adversarial Policies

5.1.1 Experiment Design. We first evaluate the winning rates of adversarial policies trained using the proposed curiosity-driven approach. We select the most relevant work [20] as the baseline, which is also the current state-of-the-art approach. Guo et al. selected 4 environments from MuJoCo [5] to evaluate the performance in zero-sum games. We choose an additional environment (`Run-To-Goal-Ants`) to further evaluate adversarial policies in low-dimensional environments. For StarCraft II, we select the default environment as displayed in the open-source repository of the baseline approach. We use the same victim agents to ensure a fair comparison. We use the PPO algorithm to train adversarial policies for both our method and the baseline. Some hyper-parameters are shared across our method and the baseline [20], e.g., discount factor, learning rate, and the number of training steps. Besides, as the network architectures may affect the ability to learn effective policies and value functions, we also adopt the same network architectures from the released repository of the baseline [20]. Details of implementation (e.g., the network structure of policies, hyper-parameter settings, etc.) are provided in Supplementary A.

5.1.2 Result Analysis. We record the winning rates of adversarial policies during the training process and plot the results in Figure 3. In this figure, the curves marked using "×" and "•" illustrate the winning rate trends of our curiosity-driven approach and the baseline, respectively. The figure shows that among all the six investigated environments, our curiosity-driven and victim-aware method outperforms the state-of-the-art baseline on three of them in terms of winning rates. Our method achieves a similar level of performance in the remaining environments.

Table 1: The non-loss rates of victim agents against three types of agents: (1) our adversarial policies (2) baseline adversarial policies and (3) regular agents. "Before" and "After" indicate the results for victims before and after adversarially training using the proposed method. Environment names are aggregated to save space. For example, `S-A` and `K-A-D` refer to `Sumo-Ants` and `Kick-And-Defend`, respectively.

Games	Ours (%)		Baseline (%)		Regular (%)	
	Before	After	Before	After	Before	After
K-A-D	16.0	51.0	23.0	41.0	48.0	30.0
Y-S-N-P	18.0	76.0	28.0	49.0	55.0	37.0
S-A	89.0	89.0	94.0	96.0	55.0	50.0
S-H	58.0	75.0	57.0	56.0	67.0	41.0
R-T-G-A	70.0	72.0	97.0	96.0	58.0	59.0
SC II	3.0	76.0	2.0	79.0	68.0	94.0

More specifically, in MuJoCo, the following high-dimensional environments where the trained agents are humanoid robots (i.e., `Kick-and-Defend`, `You-Shall-not-Pass`, and `Sumo-Humans`), are considered easier to attack, as the dimension of observation space that an adversary can manipulate is high [15]. The baseline achieved decent performance in the three environments, and results show that using our method can improve the performance of adversarial policies. In `Kick-and-Defend`, our method produces adversarial policies with an 8% higher winning rate than the baseline approach. In `You-Shall-not-Pass`, we observe an improvement of 10%. In `Sumo-Humans`, our method maintains a similar level of performance. Overall, the results demonstrate the benefits of better exploring vulnerabilities using the victim-aware curiosity mechanism.

As reported by [15] and [20], it is challenging to affect the victim's observation in low-dimensional environments, so prior adversarial attacks barely work in `Sumo-Ants` and `Run-To-Goal-Ants` environments. In `Sumo-Ants`, the winning rates of the two methods are very close. This is because in `Sumo-Ants` an agent can choose to leave the arena to achieve a draw. So adversarial policies learn to achieve a draw rather than risk the high probability of loss in playing with the victim. But in `Run-To-Goal-Ants`, where the above strategy does not work, our results show that the victim-aware curiosity mechanism finds stronger adversarial policies. The state-of-the-art method only achieves an average winning rate of 3%, while our adversarial policy boosts the average winning rate to 27%. The performance in StarCraft II is already very high for the baseline (a winning rate of 96%). Our method achieves the same winning rate as the baseline. More importantly, as indicated by the curves in Figure 3, the winning rates of adversarial policies trained using our method increase faster than that trained by the baseline, indicating that our method can better attack the victim.

5.2 Adversarial Training

5.2.1 Experiment Design. *Adversarial training* is an effective way to protect DNN against adversarial attacks [18]. In this experiment,

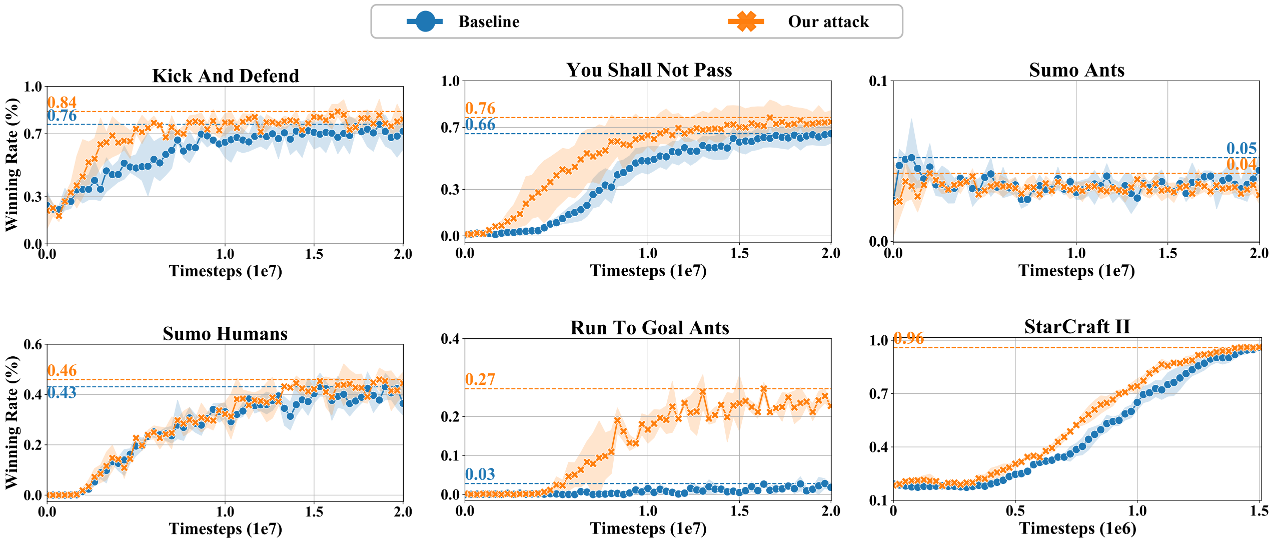


Figure 3: The winning rates of adversarial policies in six environments. Orange curves marked using "×" and blue curves marked with "•" illustrate winning rate trends of our approach and the baseline approach. The exponential moving average of each curve is set to 20 steps. Besides, we run five random seeds for both methods. The highlighted numbers on y-axis are the average winning rates of the best-performing adversarial policy. See Figure 7 in Supplementary B.1 for results of non-loss rates.

we retrain a victim against a fixed adversarial policy (best one in Section 5.1), which is called *adversarial training* of DRL [15, 20]. We analyze the effectiveness of adversarial training from the following three perspectives. *First*, we let the victim, before and after adversarial training, play against the same adversarial policy it is retrained on (in adversarial training) and collect the changes in the non-loss rate. It helps us understand whether victims can gain robustness from adversarial training. *Second*, we evaluate the adversarially trained victim against the adversarial policies from the baseline approach [20] to analyze whether the gained robustness can generalize to defend against other adversarial policies. *Third*, prior research works show that adversarial training can harm model performance on benign datasets [18], so we test how the retrained victim performs against a regular opponent agent. It is noticed that the hyper-parameter setting of adversarial retraining is the same as that we use in the adversarial policy training experiments.

5.2.2 Result Analysis. We present the performance of victims before and after adversarial training in Table 1. We first observe that adversarial training increases the victim robustness in high-dimensional environments (the "Ours" column in the table) like `You-Shall-Not-Pass`; the non-loss rate increases from 18% to 76%. Similarly, in `Kick-And-Defend`, the non-loss rate increases from 16% to 51%. However, adversarial training barely works in low-dimensional environments of MuJoCo like `Sumo-Ants`, where the change in the already high non-loss rates is negligible. We observe that the improvement in robustness from adversarial training using our adversarial policies generalizes to the robustness against the baseline method ("Baseline" column). We find that adversarial training hurts model performance on regular agents, especially in high-dimensional environments ("Regular" column). However, we note one exception that adversarial training improves the victim's performance against the regular agent in StarCraft II. Figure 4 presents the winning rates of the victim during the retraining process, indicating that adversarial training helps defend adversarial policies. Figure 8 in Supplementary B.2 shows how victims' non-loss rates change during the adversarial training process.

Table 2: The winning rates of adversarial policies are trained using (1) the baseline method, (2) the victim-*un*aware method and (3) the victim-aware method. The numbers inside the parentheses are the absolute improvement to the non-loss rates of the baseline method.

Games	Baseline (%)	Victim-*unaware* (%)	Victim-aware (%)
`Kick-And-Defend`	76.0	79.0 **(+ 3.0)**	84.0 **(+ 8.0)**
`You-Shall-Not-Pass`	66.0	71.0 **(+ 5.0)**	76.0 **(+ 10.0)**
`Sumo-Ants`	5.0	4.0 **(- 1.0)**	4.0 **(- 1.0)**
`Sumo-Humans`	43.0	44.0 **(+ 1.0)**	46.0 **(+ 3.0)**
`Run-To-Goal-Ants`	3.0	10.0 **(+ 7.0)**	27.0 **(+ 24.0)**
`StarCraft II`	96.0	96.0 **(+ 0.0)**	96.0 **(+ 0.0)**

6 ADDITIONAL ANALYSIS

We conduct additional experiments to further shed light on the strength of our proposed method. We perform an ablation study to analyze the benefits of the victim-aware exploration. We also investigate a simple defensive strategy called observation masking, as well as how the victims' policy networks are activated when playing with different types of attackers.

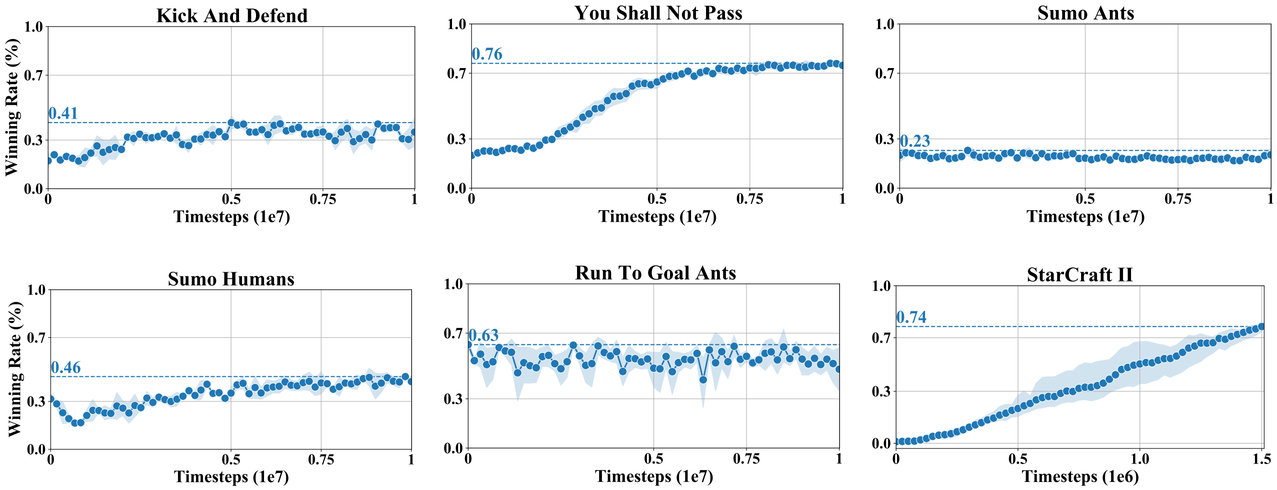


Figure 4: The winning rates of the victim agents after being retrained using our attack method play against the adversarial policies trained by our method in six games. See Figure 8 in Supplementary B.2 for results of non-loss rates.

6.1 Ablation Study

We perform this ablation study to examine whether utilizing approximated victim information helps in training better adversarial policies. We experiment with an adjusted approach in which the input to RND is changed to the raw observation rather than the output of the hidden layer of the approximated victim value network as in our main approach. In other words, the adjusted approach is no longer "victim-aware." We train adversarial policies using the adjusted approach (keep other settings the same as our main experiment in Section 5). The results are illustrated in Table 2. We observe that **the victim-aware exploration leads to stronger adversarial policies**. For instance, using the adjusted approach, the average winning rate decreases by 5% and 17% in You-Shall-Not-Pass and Run-To-Goal-Ants environment, and by 5% in Kick-And-Defend. But the adjusted approach still outperforms the baseline. The victim-aware exploration improves the averaged performance by 5.8% compared to the victim-**un**aware method. These results demonstrate that our proposed method does not improve the state-of-the-art approach solely using the curiosity mechanism; the targeted exploration of the victim's vulnerability using the output of hidden layers also contributes.

6.2 Observation Masking

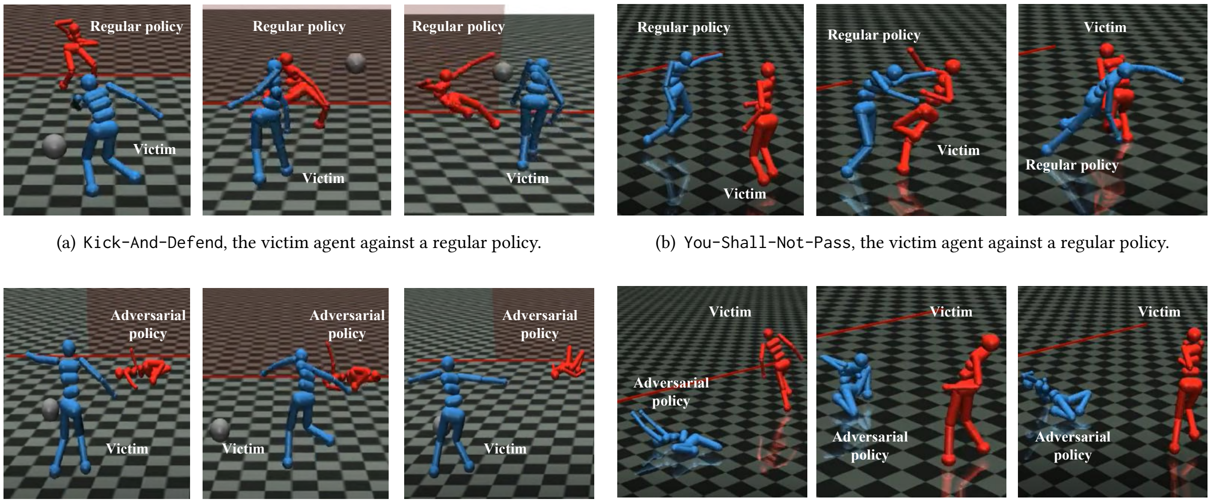
Both [15] and [20] notice that the victim can achieve a higher winning rate by simply masking the victim's partial observation of the adversary's action. Specifically, the victim's observation of the adversarial policy's actions is set as static values during the interactions. To further validate their findings, we select the best-performing adversaries from both our method and the baseline. Then we let these adversarial policies play with corresponding victims for 100 rounds. We compute the average non-loss rates of victims under two different settings, one where the observation is masked and the other where it is not. Table 3 shows the results. We confirm the finding that masking observation can defend against adversarial policies, suggesting that the adversarial policy succeeds by manipulating the victim's observations. One exception is Sumo-Ants, where the non-loss rates even before the masking are rather high. This indicates that our adversarial policies do not win by taking generally good actions (e.g., win by pushing the opponent in a sumo game); they take actions that the victim is not familiar with (e.g., lie down in a strange manner) to fool victims. But from Table 3 we find that even when observation is masked, the adversarial policies trained by our proposed approach still outperform the baseline.

Table 3: The non-loss rates of the original victim agents against (1) our adversarial policies and (2) baseline adversarial policies. "Before" and "After" indicates the results before and after masking victims' observation, respectively.

Games	Ours (%)		Baseline (%)	
	Before	After	Before	After
Kick-And-Defend	16.0	90.0	23.0	94.0
You-Shall-Not-Pass	18.0	40.0	28.0	99.0
Sumo-Ants	89.0	91.0	94.0	93.0
Sumo-Humans	58.0	70.0	57.0	76.0
Run-To-Goal-Ants	70.0	73.0	97.0	99.0
StarCraft II	3.0	14.0	2.0	18.0

6.3 Video Analysis

To more intuitively compare the difference between adversarial and regular policies, we let the victims interact with adversarial policies and regular agents and record videos demonstrating the interactions. Figure 5 illustrates some snapshots of the videos in Kick-and-Defend and You-Shall-Not-Pass games. We find that the behaviours of adversarial policies and regular agents are rather different. Specifically, we can observe that the regular agents learn



(a) `Kick-And-Defend`, the victim agent against a regular policy.

(b) `You-Shall-Not-Pass`, the victim agent against a regular policy.

(c) `Kick-And-Defend`, the victim agent against our adversarial policy.

(d) `You-Shall-Not-Pass`, the victim agent against our adversarial policy.

Figure 5: Illustrative snapshots of a victim against regular agents and our adversarial agents in `Kick-And-Defend` and `You-Shall-Not-Pass`. The victim wins if it shoots the ball to the grey region and crosses the finish line; otherwise, the attack wins. Compared with the behaviours of regular agents learning to run, kick or block, the adversarial agents never stand up and lie under the ground in some strange posts, but still win the games.

to take actions like run, push or block (as presented in Figure (a) and (b)), which are desired to win the game. However, we observe that the agents that adopt adversarial policies exhibit strange behaviours. For example, in Figure 5 (c) and (d), the attackers lie on the ground in a weird manner. Surprisingly, such actions, which create opportunities for the opponent to win, can make the victim perform poorly (e.g., fail to kick the ball or fail to cross the line). More demo videos are released in the "./MuJoCo/video" folder of our open-source repository.[1]

6.4 Policy Network Activation Analysis

We let the victims play against different types of opponents: adversarial policies trained by our method and the baseline, as well as regular agents. We record how the victims' policy networks are activated, e.g., what actions are taken. To ensure a fair comparison with the previous work [20], we let a victim play with an opponent for the same number of episodes (i.e., 250), each yielding a trajectory of 5,000 time steps. All the actions in a trajectory is viewed as an *action vector*, and then we use t-SNE [47] to visualize them in a two-dimension space. We leverage t-SNE to visualize action vectors generated by interacting with different opponents in the same figure. Due to the limited space, Figure 6 only plots the results in `Run-To-Goal-Ants`. We refer interested readers to Section B.3 of Supplementary for the full results. We can observe that the victim's network activation distributes differently when the victim faces different types of opponents. The adversarial policies can make the victim exhibit actions (i.e., red and blue dots) that distribute differently from the actions when playing with regular agents (yellow dots). Besides, the distribution of activation is also different when the victim plays with the baseline and our method: our method makes the victim's network activation (red dots in Figure 6) cover more space, suggesting that our method finds additional vulnerabilities on top of the baseline. Visualizations of other environments are presented in Figure 9 in the supplementary materials.

7 RELATED WORK

There is a series of work on testing and improving the quality of various DNN-based systems, including speech recognition [2, 3], sentiment analysis [4, 54, 56], and so on. We refer interested readers to a comprehensive survey on AI testing [63]. This section briefly discusses works related to (1) adversarial attack for deep reinforcement learning and (2) curiosity mechanisms that encourage agents to perform exploration.

7.1 Attack for DRL

Although DRL has demonstrated excellent performance in many tasks such as autonomous driving [13, 27, 59] and video game playing [16, 17, 24], it faces a series of attacks, including backdoors [26, 48, 49, 55], privacy-stealing attacks [10, 34], adversarial attack [25, 28, 38, 62], and so on.

This paper explores adversarial policy training, which falls into the category of adversarial attack. Inspired by adversarial attacks on neural networks [9, 18, 35], researchers have elicited sub-optimal actions from well-trained agents' by adding perturbations to agent observations while such perturbations do not affect decisions from the human perspective. According to the information available for searching such perturbations, adversarial attacks can be divided into two types: *white-box* and *black-box*. The white-box attack typically leverages the gradient information of DRL agents [25, 28, 29, 38, 62], while the black-box attack only has limited access to the victim models, e.g., only outputs of an agent given certain inputs [52]. The gradient information of victim models is not available in adversarial

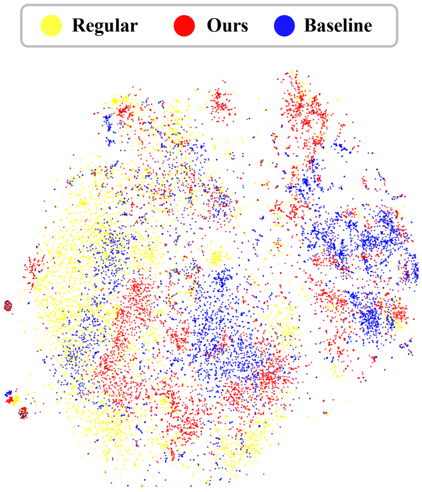


Figure 6: The t-SNE visualizations of the distribution of the victim's network activation in the `Run-To-Goal-Ant` **environment. The victim plays against different opponents: (1) our adversarial policies, (2) baseline adversarial policies, and (3) regular agents. The activation triggered by our adversarial policies distributes differently from the other two, indicating that our method uncovers different vulnerabilities.**

settings. For example, it is reasonable to assume that an attack does not know the policy network parameters of a remotely deployed agent; thus, we perform adversarial attacks in a black-box manner.

Some adversarial attack methods, even the black-box ones, are not considered feasible in practice. The reason is that adding noise to the background image typically requires an attacker to hack into the game engine first, which itself is rather difficult and even infeasible. For more practically executable attacks, researchers propose to attack victim agents by taking legal actions in the game environment. Gleave et al. [15] explore scenarios where two players (adversary and victim) compete with each other. The adversarial agent is trained to take actions allowed in the environment that can fool the victim agent into sub-optimal responses. To train such adversarial agents, a two-player game is first converted into a single-agent learning problem by fixing the victim's policy, and then PPO [40] (one of the most popular DRL algorithms) is used to train attackers to learn policies that can maximize their expected rewards. Guo et al. [20] further relax the requirement that the game should be zero-sum and design a new objective for the attacker to also minimize the victim's expected rewards. Wu et al. [51] use an explainable artificial intelligence technique to guide an attacker to take the action that the victim agent pays attention to and enhance the effectiveness of attacks. Experiments show that adversarial agents trained using the new objective [20] is the current state-of-the-art method, which is selected as the baseline in our experiment.

7.2 Curiosity Mechanism in DRL

Researchers have proposed the *curiosity mechanism* to encourage agents to explore the environment. Generally speaking, it constructs an intrinsic reward to quantify the "novelty" of a new experience [7, 8, 36, 42, 61, 66]. We summarize recent related works about the curiosity mechanism in reinforcement learning.

Count-based exploration methods record how many times a state is visited during the training process, which is a straightforward and effective way to model the distribution of visited and un-visited states [14, 30, 45]. However, count-based methods have been shown to be ineffective in environments where the state space is high-dimensional or continuous (e.g., Starcraft II) [36]. Another class of exploration methods relies on the errors in predicting dynamics, which address the difficulty in complex environments [8, 36, 37]. Pathak et al. propose the Intrinsic Curiosity Module (ICM), a model predicting the next state given the current state and action. The error between the predicted and true next state is used as an *intrinsic reward* [36]. Based on the observation that neural networks tend to have lower prediction errors on examples similar to training examples, [8] proposes Random Network Distillation (RND) that utilizes the prediction errors of networks trained on historical trajectories to quantify the novelty of states, which is effective and easy to implement in reality, e.g., generating test cases for Cyber-Physical Systems [64]. The results inspire us to use RND to encourage adversarial policies to find more vulnerabilities of victims. We leave the utilization of other exploration methods to future work.

8 CONCLUSION AND FUTURE WORK

This paper presents a novel curiosity-driven and victim-aware approach to attack victim agents in two-player games; the obtained adversarial policies outperform the current state-of-the-art results. To be victim-aware, the proposed approach approximates the state-value function of the victim in a purely black-box manner and extracts the victim information by utilizing outputs of a hidden layer in the approximated network. To be curiosity-driven, the approach applies RND to explore the hidden layer outputs to find adversarial policies more efficiently. The experiment results show that the average winning rate of adversarial policies trained using our method is higher than that of the baseline. Our method is also the first to work effectively in low-dimensional environments in MuJoCo's two-player games. An ablation study demonstrates that being aware of the approximated victim information helps in improving the performance of adversarial policies on top of the benefits of curiosity-driven exploration. These discoveries show that our attack can learn the curiosity-driven and victim-aware policies effective in defeating victims.

In the future, we plan to explore adversarial policies in multi-agent environments, the complexity of which exponentially increases with the number of players. Training adversarial policies in such complex environments is not currently supported in the current state-of-the-art. We also plan to develop more effective techniques to defend against adversarial policies.

ACKNOWLEDGMENTS

This research/project is supported by the National Research Foundation, Singapore and DSO National Laboratories under the AI Singapore Programme (AISG Award No: AISG2-RP-2020-017), the National Key Research and Development Program of China (No. 2021YFC2800501), and the National Foundation Fund of China (Y2009-1B-02-353).

REFERENCES

[1] Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Leonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. 2021. What Matters for On-Policy Deep Actor-Critic Methods? A Large-Scale Study. In *ICLR*.

[2] Muhammad Hilmi Asyrofi, Zhou Yang, and David Lo. 2021. CrossASR++: A Modular Differential Testing Framework for Automatic Speech Recognition. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Athens, Greece) *(ESEC/FSE 2021)*. Association for Computing Machinery, New York, NY, USA, 1575–1579. https://doi.org/10.1145/3468264.3473124

[3] Muhammad Hilmi Asyrofi, Zhou Yang, Jieke Shi, Chu Wei Quan, and David Lo. 2021. Can Differential Testing Improve Automatic Speech Recognition Systems?. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 674–678. https://doi.org/10.1109/ICSME52107.2021.00079

[4] Muhammad Hilmi Asyrofi, Zhou Yang, Imam Nur Bani Yusuf, Hong Jin Kang, Ferdian Thung, and David Lo. 2021. BiasFinder: Metamorphic Test Generation to Uncover Bias for Sentiment Analysis Systems. *IEEE Transactions on Software Engineering* (2021), 1–1. https://doi.org/10.1109/TSE.2021.3136169

[5] Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. 2018. Emergent Complexity via Multi-Agent Competition. In *ICLR*.

[6] Vahid Behzadan and Arslan Munir. 2017. Vulnerability of deep reinforcement learning to policy induction attacks. In *International Conference on Machine Learning and Data Mining in Pattern Recognition*. 262–275.

[7] Yuri Burda, Harri Edwards, Deepak Pathak, et al. 2018. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355* (2018).

[8] Yuri Burda, Harrison Edwards, Amos Storkey, et al. 2018. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894* (2018).

[9] Nicholas Carlini and David Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. In *2017 IEEE Symposium on Security and Privacy (SP)*. 39–57.

[10] Kangjie Chen, Shangwei Guo, Tianwei Zhang, Xiaofei Xie, and Yang Liu. 2021. Stealing Deep Reinforcement Learning Models for Fun and Profit. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security (ASIA CCS '21)*. Association for Computing Machinery, New York, NY, USA, 307–319.

[11] Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, et al. 2022. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature* 602, 7897 (2022), 414–419.

[12] Richard Elderman., Leon J. J. Pater., Albert S. Thie., Madalina M. Drugan., and Marco M. Wiering. 2017. Adversarial Reinforcement Learning in a Cyber Security Simulation. In *Proceedings of the 9th International Conference on Agents and Artificial Intelligence*. SciTePress, 559–566. https://doi.org/10.5220/0006197105590566

[13] Abdur R. Fayjie, Sabir Hossain, Doukhi Oualid, and Deok-Jin Lee. 2018. Driverless Car: Autonomous Driving Using Deep Reinforcement Learning in Urban Environment. In *15th International Conference on Ubiquitous Robots (UR)*. 896–901.

[14] Lior Fox, Leshem Choshen, and Yonatan Loewenstein. 2018. DORA The Explorer: Directed Outreaching Reinforcement Action-Selection. In *ICLR*.

[15] Adam Gleave, Michael Dennis, Cody Wild, et al. 2020. Adversarial Policies: Attacking Deep Reinforcement Learning. In *ICLR*.

[16] Chen Gong, Yunpeng Bai, Xinwen Hou, et al. 2020. Stable Training of Bellman Error in Reinforcement Learning. In *Neural Information Processing*. 439–448.

[17] Chen Gong, Qiang He, Yunpeng Bai, et al. 2021. Wide-Sense Stationary Policy Optimization with Bellman Residual on Video Games. In *2021 IEEE International Conference on Multimedia and Expo (ICME)*. 1–6.

[18] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *ICLR*. http://arxiv.org/abs/1412.6572

[19] Shixiang Gu, Ethan Holly, Timothy Lillicrap, et al. 2017. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *ICRA*. IEEE, 3389–3396.

[20] Wenbo Guo, Xian Wu, Sui Huang, and Xinyu Xing. 2021. Adversarial Policy Learning in Two-player Competitive Games. In *Proceedings of the 38th International Conference on Machine Learning*, Vol. 139. PMLR, 3910–3919.

[21] Kim Hammar and Rolf Stadler. 2020. Finding Effective Security Strategies through Reinforcement Learning and Self-Play. In *2020 16th International Conference on Network and Service Management (CNSM)*. 1–9.

[22] Kim Hammar and Rolf Stadler. 2022. Learning Security Strategies through Game Play and Optimal Stopping. *CoRR* abs/2205.14694 (2022). arXiv:2205.14694

[23] Qiang He, Chen Gong, Yuxun Qu, Xiaoyu Chen, Xinwen Hou, and Yu Liu. 2021. Mepg: A minimalist ensemble policy gradient framework for deep reinforcement learning. *arXiv preprint arXiv:2109.10552* (2021).

[24] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, et al. 2018. Rainbow: Combining Improvements in Deep Reinforcement Learning. In *AAAI*. 3215–3222.

[25] Sandy H. Huang, Nicolas Papernot, Ian J. Goodfellow, Yan Duan, and Pieter Abbeel. 2017. Adversarial Attacks on Neural Network Policies. In *5th International Conference on Learning Representations, ICLR 2017, Workshop Track Proceedings*.

[26] Panagiota Kiourti, Kacper Wardega, Susmit Jha, and Wenchao Li. 2020. TrojDRL: evaluation of backdoor attacks on deep reinforcement learning. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[27] Bangalore Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Kumar Yogamani, and Patrick Pérez. 2020. Deep Reinforcement Learning for Autonomous Driving: A Survey. abs/2002.00444 (2020).

[28] Jernej Kos and Dawn Song. 2017. Delving into adversarial attacks on deep policies. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*.

[29] Yenchen Lin, Zhangwei Hong, Yuanhong Liao, et al. 2017. Tactics of Adversarial Attack on Deep Reinforcement Learning Agents. In *IJCAI*. 3756–3762.

[30] Marlos C. Machado, Marc G. Bellemare, and Michael Bowling. 2020. Count-Based Exploration with the Successor Representation. *Proceedings of the AAAI Conference on Artificial Intelligence* 34, 04 (2020), 5125–5133.

[31] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, et al. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *ICLR*.

[32] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[33] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, et al. 2019. Solving Rubik's Cube with a Robot Hand. *CoRR* abs/1910.07113 (2019). arXiv:1910.07113

[34] Xinlei Pan, Weiyao Wang, Xiaoshuai Zhang, et al. 2019. How You Act Tells a Lot: Privacy-Leaking Attack on Deep Reinforcement Learning. In *the International Conference on Autonomous Agents and MultiAgent Systems*. 368–376.

[35] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, et al. 2017. Practical Black-Box Attacks against Machine Learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. 506–519.

[36] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. 2017. Curiosity-driven Exploration by Self-supervised Prediction. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 70)*. PMLR, 2778–2787.

[37] Roberta Raileanu and Tim Rocktäschel. 2020. RIDE: Rewarding Impact-Driven Exploration for Procedurally-Generated Environments. In *ICLR*.

[38] Alessio Russo and Alexandre Proutière. 2019. Optimal Attacks on Reinforcement Learning Policies. *CoRR* abs/1907.13548 (2019). http://arxiv.org/abs/1907.13548

[39] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, et al. 2017. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging* 2017, 19 (2017), 70–76.

[40] John Schulman, Filip Wolski, Prafulla Dhariwal, et al. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[41] David Silver, Thomas Hubert, Julian Schrittwieser, et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144.

[42] Chenyu Sun, Hangwei Qian, and Chunyan Miao. 2022. CCLF: A Contrastive-Curiosity-Driven Learning Framework for Sample-Efficient Reinforcement Learning. *arXiv preprint arXiv:2205.00943* (2022).

[43] Peng Sun, Xinghai Sun, Lei Han, Jiechao Xiong, Qing Wang, Bo Li, Yang Zheng, Ji Liu, Yongsheng Liu, Han Liu, and Tong Zhang. 2018. TStarBots: Defeating the Cheating Level Builtin AI in StarCraft II in the Full Game. arXiv:1809.07193 [cs.AI]

[44] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement learning: An introduction.* MIT press.

[45] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. 2017. #Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning. In *Advances in Neural Information Processing Systems*, Vol. 30.

[46] Gerald Tesauro. 1995. Temporal difference learning and TD-Gammon. *Commun. ACM* 38, 3 (1995), 58–68.

[47] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, 11 (2008).

[48] Lun Wang, Zaynah Javed, Xian Wu, Wenbo Guo, Xinyu Xing, and Dawn Song. 2021. BACKDOORL: Backdoor Attack against Competitive Reinforcement Learning. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. 3699–3705.

[49] Yue Wang, Esha Sarkar, Wenqing Li, Michail Maniatakos, and Saif Eddin Jabari. 2021. Stop-and-go: Exploring backdoor attacks on deep reinforcement learning-based traffic congestion control systems. *IEEE Transactions on Information Forensics and Security* 16 (2021), 4772–4787.

[50] Yufei Wang, Zheyuan Ryan Shi, Lantao Yu, et al. 2019. Deep reinforcement learning for green security games with real-time information. In *the Association for the Advancement of Artificial Intelligence,* AAAI, Vol. 33. 1401–1408.

[51] Xian Wu, Wenbo Guo, Hua Wei, and Xinyu Xing. 2021. Adversarial Policy Training against Deep Reinforcement Learning. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 1883–1900.

[52] Chaowei Xiao, Xinlei Pan, Warren He, et al. 2019. Characterizing attacks on deep reinforcement learning. *arXiv preprint arXiv:1907.09470* (2019).

[53] Zhiwei Xu, Bin Zhang, Yunpeng Bai, et al. 2021. Learning to Coordinate via Multiple Graph Neural Networks. In *Neural Information Processing*. 52–63.

[54] Zhou Yang, Muhammad Hilmi Asyrofi, and David Lo. 2021. BiasRV: Uncovering Biased Sentiment Predictions at Runtime. In *Proceedings of the 29th*

ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (Athens, Greece) *(ESEC/FSE 2021)*. Association for Computing Machinery, New York, NY, USA, 1540–1544. https://doi.org/10.1145/3468264.3473117

[55] Zhaoyuan Yang, Naresh Iyer, Johan Reimann, and Nurali Virani. 2019. Design of intentional backdoors in sequential models. *arXiv:1902.09972* (2019).

[56] Zhou Yang, Harshit Jain, Jieke Shi, Muhammad Hilmi Asyrofi, and David Lo. 2021. BiasHeal: On-the-Fly Black-Box Healing of Bias in Sentiment Analysis Systems. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 644–648. https://doi.org/10.1109/ICSME52107.2021.00073

[57] Zhou Yang, Jieke Shi, Muhammad Hilmi Asyrofi, and David Lo. 2022. Revisiting Neuron Coverage Metrics and Quality of Deep Neural Networks. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE Computer Society.

[58] Zhou Yang, Jieke Shi, Junda He, and David Lo. 2022. Natural Attack for Pre-Trained Models of Code. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 1482–1493.

[59] Changkun Ye, Huimin Ma, Xiaoqin Zhang, Kai Zhang, and Shaodi You. 2017. Survival-Oriented Reinforcement Learning Model: An Effcient and Robust Deep Reinforcement Learning Algorithm for Autonomous Driving Problem, Yao Zhao, Xiangwei Kong, and David Taubman (Eds.). 417–429.

[60] Yingjun Ye, Xiaohui Zhang, and Jian Sun. 2019. Automated vehicle's behavior decision making using deep reinforcement learning and high-fidelity simulation environment. *Transportation Research Part C: Emerging Technologies* 107 (2019), 155–170.

[61] Chuheng Zhang, Yuanying Cai, Longbo Huang, and Jian Li. 2021. Exploration by Maximizing Renyi Entropy for Reward-Free RL Framework. *Proceedings of the AAAI Conference on Artificial Intelligence* 35, 12 (2021), 10859–10867.

[62] Huan Zhang, Hongge Chen, Duane Boning, et al. 2021. Robust Reinforcement Learning on State Observations with Learned Optimal Adversary.

[63] Jie M. Zhang, Mark Harman, Lei Ma, and Yang Liu. 2022. Machine Learning Testing: Survey, Landscapes and Horizons. *IEEE Transactions on Software Engineering* 48, 1 (2022), 1–36. https://doi.org/10.1109/TSE.2019.2962027

[64] Shaohua Zhang, Shuang Liu, Jun Sun, et al. 2021. FIGCPS: Effective Failure-inducing Input Generation for Cyber-Physical Systems with Deep Reinforcement Learning. In *36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 555–567.

[65] Yiren Zhao, Ilia Shumailov, and Han Cui. 2020. Blackbox Attacks on Reinforcement Learning Agents Using Approximated Temporal Information. In *International Conference on Dependable Systems and Networks Workshops*. 16–24.

[66] Lulu Zheng, Jiarui Chen, Jianhao Wang, Jiamin He, Yujing Hu, Yingfeng Chen, Changjie Fan, Yang Gao, and Chongjie Zhang. 2021. Episodic Multi-agent Reinforcement Learning with Curiosity-driven Exploration. In *Advances in Neural Information Processing Systems*, Vol. 34. 3757–3769.

Supplementary Materials

A EXPERIMENTAL DETAILS

A.1 Implementation and Hyper-parameters

The implementation of both our method and the baseline are based on `TensorFlow`[2] framework and `Stable-Baselines`[3] libraries. We retrieve the implementation for the baseline attack [20] from its official Github repository released by the authors[4]. Besides, the game environment wrappers are implemented based on the OpenAI Gym[5], and the PySC2 extension packages[6].

Some hyper-parameters are shared across our method and the baseline [20], e.g., discount factor γ, learning rate ϵ, and the number of steps n. To ensure a fair comparison, we set these hyper-parameters as the same for both methods by using the hyper-parameters as shown in the baseline repository[4]. Besides, as the network architectures may affect the ability to learn effective policies and value functions, we also adopt the same network architectures from the repository.

Both the networks of the adversarial policy and the state-value function take the raw observation as input. The network of adversarial policy produces the probability distribution of actions. The network of state-value function produces a single value indicating the state value. We take the *penultimate* hidden layer of the approximated victim state-value function network and feed the outputs of this hidden layer into the RND networks. Intuitively speaking, a state-value function network can be viewed as feature extractor layers followed by layers to process the output of the feature extractor; as a consequence the penultimate hidden layer contains the most compact and relevant information needed for producing the value output. It might help to additionally use other low-level hidden layers' output, however, our experiments reveal that our approach using only the penultimate layer is sufficient to beat the state of the art [20]. The size of the inputs to RND is the same as the output size of the hidden layer: 256 in all environments. The predictor network and the target network share the same network architecture, both of which have 2 layers. The parameters of the two networks are randomly initialized, but we fix the target network and only allow the predictor network to learn the output of target network.

In the experiments of adversarial training, we retrain the policy network of the original victim agents. The architecture of the policy network as well as the state-value function, and the training hyper-parameters (e.g., number of training epochs, learning rate, the discount factor, etc.) are the same as that in adversarial policy training experiments. When training the adversarial policies, it is noted that compared to the baseline, we introduce an important hyper-parameter to our curiosity-driven and victim-aware approach: the exploration coefficient λ, i.e., the weight to balance the advantage functions of original reward and intrinsic reward in Eqn. (8). A higher λ indicates that an agent pays more attention to exploring the environments. We choose different values of λ in different environments, which are presented in Table 4.

For the StarCraft II games, although we keep the hyper-parameter settings the same for both methods, the repository of the baseline method [20] does not indicate the specific version of StarCraft II they performed the experiments on. To ensure a fair comparison as much as we can, we select version 4.6.2 (B69232) and the Melee/Flat64.SC2 map, in which we can replicate their results. The details of network architectures can be found here.[7]

A.2 Experiment Setups

To evaluate the effectiveness of our proposed method, we conduct experiments on zero-sum games (MuJoCo[8]) and one non-zero-sum game (StarCraft II from PySC2), both of which are commonly adopted to evaluate reinforcement learning algorithms in two-player game contexts. For demonstrating the performance of adversarial policies, we first need to select appropriate victim agents under attack. Similar to the practice in [20], the selection is operated as follows:

(1) We obtained the pre-trained agents from the Agent Zoo package [5] and evaluate the performance of these agents against the adversarial policies in [15].

[2] https://github.com/tensorflow/tensorflow

[3] https://github.com/Stable-Baselines-Team/stable-baselines

[4] https://github.com/psuwuxian/rl_adv_valuediff

[5] https://gym.openai.com

[6] https://github.com/Tencent/PySC2TencentExtension

[7] https://github.com/Tencent/TStarBot1

[8] https://github.com/deepmind/mujoco

Table 4: The value of exploration coefficient λ in different games.

Games	Kick-And-Defend	You-Shall-Not-Pass	Sumo-Humans	Sumo-Ants	Run-To-Goal-Ants	StarCraft II
λ	0.5	1.0	0.5	0.5	1.0	0.5

Table 5: The IDs of selected victim agents and regular agents in Agent Zoo package.

Games	Victim ID	Regular ID
Kick-And-Defend	1	3
You Shall Not Pass	1	1
Sumo-Ants	1	2
Sumo-Humans	3	2
Run-To-Goal-Ants	1	1

(2) For each environment, we select the best-performing (in terms of winning rates) agent as the victim agent under attack. By doing so, we can obtain the same group of victims used in [20]. For Run-To-Goal-Ant, which is additionally included in our experiments, there is only one agent available and we use it as the victim.

(3) For each environment, we select the *second* best-performing (in terms of winning rates) agent as the regular agent used in the adversarial training experiment (Section 5.2) and the activation analysis (Section B.3).

Each agent in the Agent Zoo is labelled with an ID to be uniquely identified. We present the id of selected victim agents and regular agents we selected in Table 5. There are two roles in the asymmetric games like Kick-And-Defend. For example, Kick-And-Defend has the kicker and defender; You-Shall-Not-Pass has the runner and blocker. We select the defender and blocker as the attackers. The rest roles (i.e., the kicker and the runner) are regular agents. For the real-time strategy game StarCraft II, we directly use the victim agent and regular agent from the official Github repository released by the authors of [20]. They use the PPO algorithm to let players in a game learn by self-playing until both agents present decisive winning rates (94% in their settings) against a rule-based agent. In our experiments, we select the same victim agent and regular agent as the experiments in [20].

We describe the setup for training adversarial policies and performing adversarial training on victim agents. It is well-known that the performance of deep reinforcement learning algorithms usually suffers from problematic unstable due to the complex and uncertain nature of environments, e.g., initial states, the probabilistic state transitions, and noise in environments [36]. To alleviate the threat of instability caused by the factors stated above, we run experiments in each environment using five different random seeds and record the average results as the final results. Besides, to illustrate the variance of the algorithm performance, we also plot the performance range. For all MuJoCo environments, although [20] trains adversarial policies for 35 million steps, we train adversarial policies for 20 million steps as the winning rates trend has already plateaued at around 15 million steps. The adversarial policies are trained for 1.5 million steps for the StarCraft II environments.

Similar to the setups for training adversarial policies, when adversarially retraining the victim agents, we also restart the experiments with 5 different random seeds and record the average performance accordingly. For MuJoCo and StarCraft II games, we take 10 million and 1.5 million steps to train the victim agents, respectively, which is the same parameter settings as [20].

B EXPERIMENTS

B.1 The Performance of Adversarial Policies

In Section 5.1, we mainly explain the performance of our proposed method in terms of winning rates. Here we explain an additional metric, i.e., the non-loss rate, and further compare our method and the baseline method. The winning rates and non-loss rates of adversarial policies during the training process are presented in Figure 7. In this figure, the curves marked using '×' and '•' illustrate the winning rate and non-loss trend of our curiosity-driven method and the baseline, respectively. The figure shows that among all six investigated environments, the proposed approach outperforms the state-of-the-art baseline in three of them (in terms of both winning rates and non-loss rates) and achieves a similar level of performance in the rest environments.

In MuJoCo's high-dimensional tasks (e.g., Kick-and-Defend and You-Shall-not-Pass), the results show that our proposed adversarial policies not only improve the performance, but also accelerate the process of convergence in training phases (trends of the winning and non-loss rates plateau earlier than the baseline). In Kick-and-Defend, our method can train adversarial policies that have an 8% higher non-loss rate than the baseline method. In You-Shall-not-Pass, we observe an improvement of 10% for the non-loss rate. Note that no ties are possible in You-Shall-not-Pass, so the trend of winning rate and non-loss rate are completely the same in this game. In Sumo-Humans, our method maintains a similar level of performance.

The prior adversarial attacks barely work in low-dimensional environments (i.e., Sumo-Ants and Run-To-Goal-Ants). In Sumo-Ants, the non-loss rates of the two methods are very close. As we mentioned in Section 5.1, this is because in Sumo-Ants an agent can choose to leave the arena to achieve a draw. Therefore, adversarial policies learn to achieve a draw rather than risk the high probability of loss in playing with the victim. Although the winning rates of adversarial policies trained by previous methods in Sumo-Ants are low, it achieves a high non-loss rate (i.e., 83%). In Run-To-Goal-Ants, both the winning and non-loss rates of prior adversarial policies are rather low. Our results show that our proposed method finds more effective adversarial policies. The state-of-the-art method only achieves an average winning rate of 3%, while our adversarial policy boosts the average winning rate to 27%. The baseline can achieve a non-loss rate of 22%, but our method can boost the value to 38%. The baseline method has already performed very well in StarCraft II (with a non-loss rate of 98%). Figure 7 illustrates that

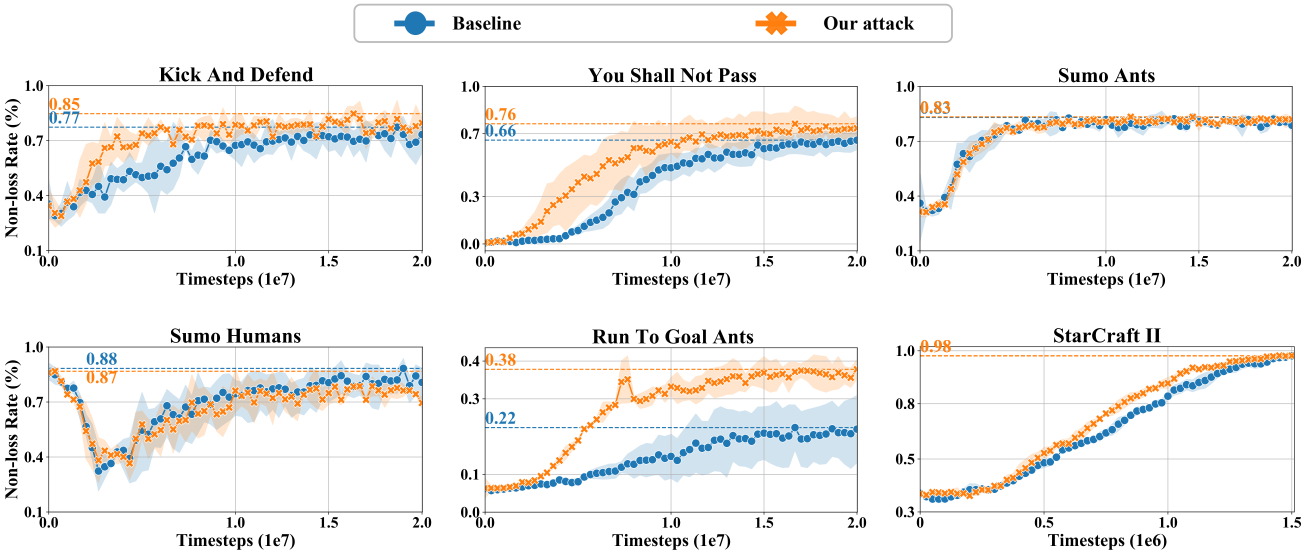


Figure 7: The non-loss rates (*i.e.,* winning rates plus tie rates) of adversarial agents in six environments. We train the adversarial policies using two methods: (1) our proposed method and (2) the baseline proposed in [20]. The exponential moving average of each curve is set to 20 steps. Besides, we run five random seeds for both methods and uses average value as the final results. The solid line denotes the mean performance (i.e., the wining rate and non-loss rates), and shadow denotes the standard deviation. The highlighted y-axis labels are the highest average non-loss rates over the total amount of iterations.

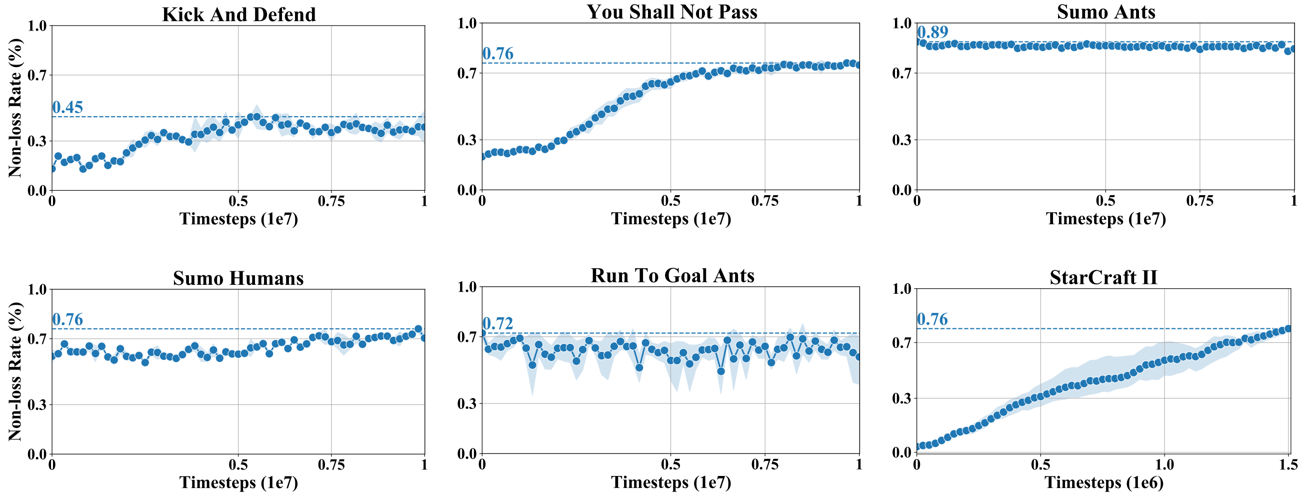
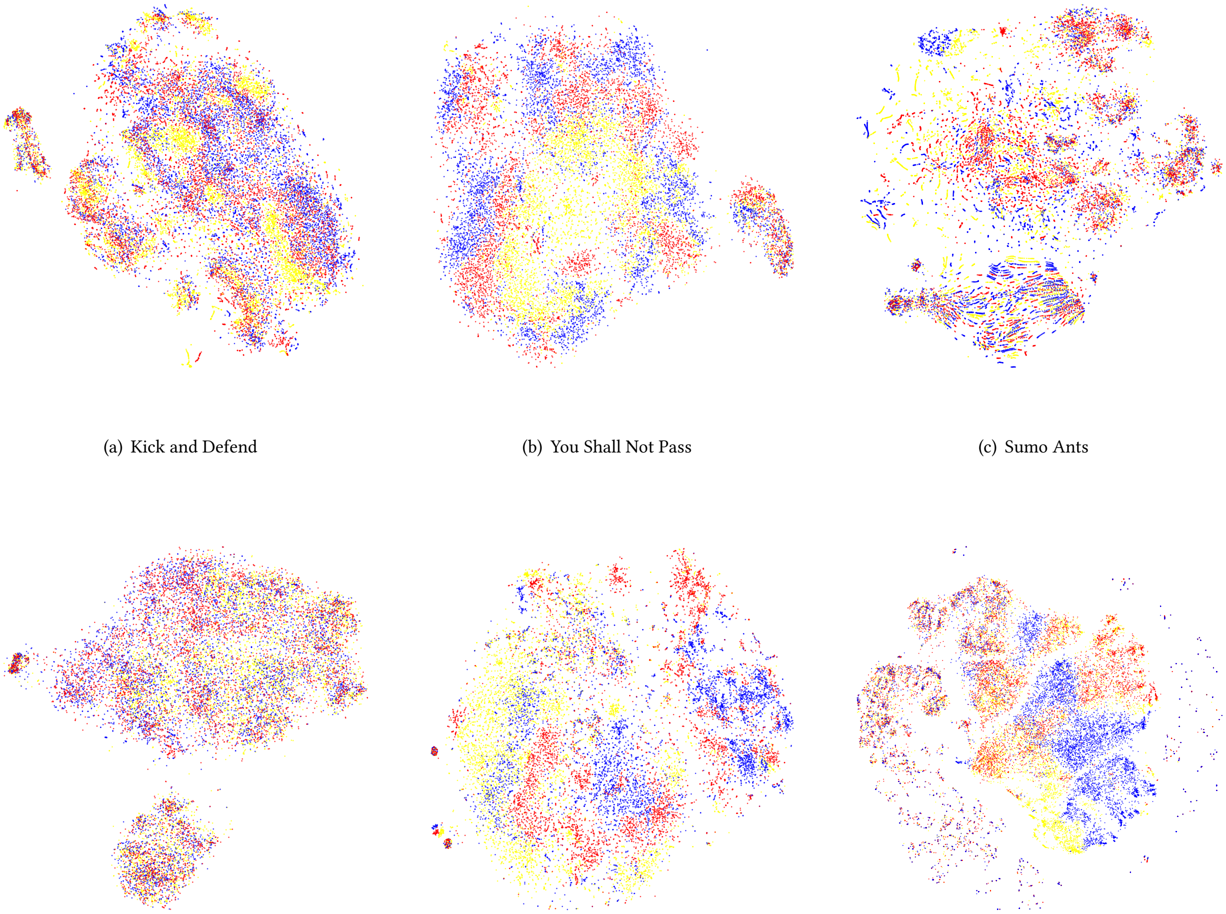


Figure 8: The non-loss rates of the retrained victim agent against with the the adversarial policy trained by our method.

our method achieves the same non-loss rate, so we only use one number (i.e., 0.98) to label the non-loss rate, but the performance of our method increases faster than the baseline.

B.2 Performances of Adversarial Training

Due to the limited space, Section 5.2 in the main paper only presents the results before and after the victims are adversarially trained, ignoring the trends of metrics during the adversarial training process. Figure 8 illustrates the trends of victims' winning rates and non-loss rates in the process of retraining, indicating that adversarial training helps enhance the victim's robustness and defend adversarial policies. This figure shows that the victims' performance improves in 4 out of 6 environments. We can also observe that the performance improvement is more significant for victims in high-dimensional environments like `You-Shall-Not-Pass`; the non-loss rate increases from 18% to 76%. Besides, in non-zero-sum environment, i.e., `StarCraft II`, the non-loss rate increases from 2% to



(a) Kick and Defend

(b) You Shall Not Pass

(c) Sumo Ants

(d) Sumo Humans

(e) Run to Goal Ants

(f) StarCraft II

Figure 9: t-SNE visualizations of the victim policy network activation pattern when playing with different opponents in MuJoCo games and StarCraft II game. The yellow dots are the victim activation when setting a regular agent as the opponent. Model fitted with a perplexity of 250 to activation from 5000 time steps against each opponent. The blue indicates the victim activation when playing against the opponents trained by the baseline method. The red dots are the activation when playing against our adversarial agent.

76%. However, adversarial training barely works in ant-against environments like `Sumo-Ants`, where the non-loss rates are already high before the retraining.

B.3 Activation Analysis

To more intuitively compare the difference between adversarial agents and regular agents, we let the original victim agents interact with different types of opponents: adversarial policies trained by our method and the baseline method, as well as regular agents. From our released demo videos in link[1], we can also observe that the adversarial policy defeats the victim by taking uncommon actions (e.g., laying down strangely) to manipulate the victim's observation rather than normal actions (e.g., pushing). Then, we record the activation of the victims' policy networks and then use t-SNE to visualize the distribution of network activation in Figure 9. It can be clearly observed that adversarial policies trained by both methods have different activation distributions from the regular agent. We also find that our method triggers the victim agent to perform actions different from activated actions when playing with a baseline method. It suggests that our method finds additional vulnerabilities on top of the baseline. These results reveal that our attack trains an adversarial policy that exploits various weaknesses than the baseline.