Adversarial Policy Learning in Two-player Competitive Games

Wenbo Guo [1] **Xian Wu** [1] **Sui Huang** [2] **Xinyu Xing** [1]

Abstract

In a two-player deep reinforcement learning task, recent work shows an attacker could learn an adversarial policy that triggers a target agent to perform poorly and even react in an undesired way. However, its efficacy heavily relies upon the zero-sum assumption made in the two-player game. In this work, we propose a new adversarial learning algorithm. It addresses the problem by resetting the optimization goal in the learning process and designing a new surrogate optimization function. Our experiments show that our method significantly improves adversarial agents' exploitability compared with the state-of-art attack. Besides, we also discover that our method could augment an agent with the ability to abuse the target game's unfairness. Finally, we show that agents adversarially retrained against our adversarial agents could obtain stronger adversary-resistance.

1. Introduction

Along with the great success of Deep Reinforcement Learning (DRL) in the application of two-player games comes a new form of attack that exploits the weakness of a policy network to fail the corresponding game agent. In the past, the development of this new kind of attack makes many unrealistic assumptions. The most common one is that an attacker has the capability of manipulating the observation of an agent by altering the environment with which the agent interacts. Under this assumption, researchers have proposed a variety of technical methods, demonstrating significant effectiveness in failing a game agent in a complicated game task (*e.g.,* (Huang et al., 2017; Lin et al., 2020)).

Sharing the similar viewpoint elaborated in (Gleave et al., 2020), we argue that attacks developed under this assumption are not practical. For example, given a master agent deployed in an online video game platform playing with hundreds of thousands of professional participants, the manipulation of the agent's observation could mean the free alternation of pixel values in the game environment (*e.g.,* changing the canvas color of the Atari game (ATARI, 2006)). In this context, to achieve this objective, an attacker might have to spend thousands of hours of effort on hacking the game platform and expect to obtain unauthorized access to the master agent and the environment with which it interacts. Given the advance of software hardening technology, such an expectation cannot always be guaranteed.

In this work, we do not bake in this assumption and propose a new attack in the context of a two-player competitive game. Like the recent research (Gleave et al., 2020), we train an adversarial agent that interacts with a well-trained victim agent in a two-player game environment and triggers it to react in an undesired fashion. Differently, as we will elaborate in Section 3.1, our learning method, however, relaxes an implicit and sometimes unrealistic assumption – a two-player game must strictly follow a zero-sum setting. We show this relaxation makes adversarial learning effective for more sophisticated games (*e.g.,* StarCraft). Besides, our learning method goes beyond exploiting the weakness of a victim agent. It demonstrates an ability to abuse the target game's unfairness when an adversarial agent has minimal impact on the victim's decision making.

Technically, instead of the straightforward adoption of an existing RL approach as is proposed in (Gleave et al., 2020), our adversarial learning method introduces a new learning algorithm that not only maximizes the expected reward of the adversarial agent but, more importantly, minimizes that of the victim. As is specified in Section 3.2, to build our learning algorithm correctly, we have to tackle a monotonicity challenge or, in other words, ensure the update of an adversarial policy does not incur the random fluctuation of the learning objective. To do this, we first remodel a two-player game and define both the adversary and victim's expected rewards as a function of the adversarial policy. Second, we carefully design a new objective function that could theoretically guarantee the monotonic property in the entire policy learning process.

[1]College of Information Sciences and Technology, The Pennsylvania State University, State College, PA, USA [2]Netflix Inc., Los Gatos, CA, USA. Correspondence to: Wenbo Guo <wzg13@ist.psu.edu>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

With all these technical endeavors above, Section 4 shows that we bring fundamental advantages over the state-of-the-art technique (Gleave et al., 2020) from many aspects. First, we can enable effective, practical adversarial attacks against sophisticated games (*e.g.,* StarCraft). To the best of our knowledge, this is the first work that successfully demonstrates a practical attack against a sophisticated video game (*i.e.,* StarCraft II). Second, we can exploit the unfairness of a game to defeat a target victim agent even if the adversarial agent has minimal influence on the victim's decision making. Third, we can learn an adversarial agent with stronger exploitability and better transferability. It makes the attack more powerful and practical. Fourth, our learning algorithm overall demonstrates less performance variance in the agent learning process than the existing method. It indicates that, by using our approach for learning an adversarial policy, an attacker could better receive an agent with a consistent hostile capability. Last but not least, our adversarial agent can help agent developers learn an agent with stronger adversary-resistance. We released our source code to support future research.[1]

2. Related Work

Existing attacks on DRL policies primarily follow three methods – ❶ attack through observation manipulation, ❷ attack through action/trajectory manipulation, and ❸ attack through an adversarial agent. Below, we summarize these works and discuss their differences from ours.

Attack through observation manipulation. Following the conventional attacks on DNNs (Goodfellow et al., 2015; Papernot et al., 2016; Carlini & Wagner, 2017; Madry et al., 2018), Huang et al. (2017) and Behzadan & Munir (2017) first proposed to perturb the victim's observation at each time step, forced its policy network to output sub-optimal actions, and thus failed the corresponding task. As a follow-up, recent efforts (Kos & Song, 2017; Russo & Proutiere, 2019; Lin et al., 2017; Sun et al., 2020; Zhang et al., 2021) improved the efficiency of such attacks by manipulating the observation of a victim agent at some selected time steps rather than the entire training trajectories.

In addition to the efficiency improvement, some existing works (Huang et al., 2017; Behzadan & Munir, 2017; Zhao et al., 2019; Xiao et al., 2019; Lin et al., 2020) extended the observation manipulation attacks to black-box settings, in which attacks can access only the input and output of a victim agent's policy network or those of a Q network. In this work, we do not assume the full privilege of manipulating an agent's observations but only the control over an adversarial agent. As is discussed in Gleave et al. (2020), this relaxation makes the attack more realistic and cost-effective.

Attack through action/trajectory manipulation. Different from the observation manipulation attacks mentioned above, another line of research directly perturbs the output of the policy network or, in other words, manipulates the action taken by victim agents. Similar to the observation manipulation attacks, researchers have also demonstrated the effectiveness of action manipulation in both black-box and white-box settings (Lee et al., 2020; Xiao et al., 2019).

Inspired by the observation and action manipulation attacks launched at the testing phase, some other researchers also proposed to launch an attack at the training phase. Specifically, they demonstrated that, by manipulating the reward in training trajectories, attackers could train a victim agent failing its corresponding task (Ma et al., 2019; Lykouris et al., 2019; Yang et al., 2019; Kiourti et al., 2019). Like the statement made above, our work removes the assumption of providing an adversary with the ability to vary action or trajectories, making the attack more practical.

Attack through an adversarial agent. Unlike the two attack methods mentioned above, there is an emerging attack, focusing on training an adversarial agent to beat its opponent in a two-player game. For example, Gleave *et al.* (Gleave et al., 2020) trained an adversarial agent for a set of MuJoCo games (Todorov et al., 2012) by using the PPO algorithm. They have demonstrated that their adversarial agent could defeat its opponent agent, exhibiting a higher winning rate. In this work, our method follows an entirely different idea. It can theoretically guarantee the adversarial agent could better discover and exploit its opponent's weakness even if a two-player game does not strictly follow a zero-sum setting.

3. Proposed Technique

3.1. Problem Scope and Assumption

In this work, we fix one agent's policy (either deterministic or stochastic policy), and train the other to win the corresponding two-player Markov game. Note that this setup is the same as the one in (Gleave et al., 2020). It simulates a real-world scenario. A game vendor first releases a master RL agent. Then, an attacker trains an adversarial agent to exploit this master's weakness and win the game for fun or profit (Supplement S6 shows an initial experiment of attacking a victim that varies its policy). Under this setup, our work makes the following two assumptions. First, we relax an implicit assumption of the existing attack (Gleave et al., 2020). That is, increasing one player's reward will decrease the other player's reward gain. This is true for all zero-sum two player games. However, in most of the real-world

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

two-player games, the rewards are not designed as exactly zero-sum. One common practice of breaking the zero-sum equilibrium is to assign both agents with the same positive or negative reward when a draw occurs (Bansal et al., 2018; OpenAI, 2017). The other is to introduce intermediate rewards and bring other benefits to agents. For example, it could help agents establish natural behaviors (Bansal et al., 2018), encourage their correct movements (Unity, 2020; Sun et al., 2018), and punish their actions that break the game rules (Unity, 2020; ATARI, 2006).

Second, following the setup in (Gleave et al., 2020), we assume an adversary can only play his/her adversarial agent with the victim agent in the corresponding game environment but does not has access to the victim player's observation, value function, and policy network. To enable a stronger attack, we further assume that the attacker knows the victim's instant reward. We believe this is a reasonable and practical assumption. In a two-player game, the instant reward of each agent is determined based on the outcome of each round of the game (*e.g.,* win or loss). Besides, it is relevant to some statistics during the play (*e.g.,* the number of enemies killed and the number of collected resources in StarCraft II (Sun et al., 2018)). This information is always available for each player of the game. In Supplementary Section S7, we demonstrate the transferability of our attack. It implies that, even if instant rewards are not accessible, an attacker could still train an adversarial agent in an environment with visible instant rewards and then successfully launch attacks against the victims.

3.2. Technical Overview and Challenge

As is mentioned above, the existing attack (Gleave et al., 2020) relies on the zero-sum assumption, which implies the increase in one player's reward gain will result in the decrease in the other player's gain. Based on this assumption, by only maximizing the adversarial player's expected total reward using the PPO algorithm, this attack could search for an adversarial policy that consistently decreases the victim reward and thus fails the victim agent. However, for most of the real-world Markov games, where the rewards are not zero-sum, merely maximizing the adversarial reward cannot guarantee to reduce that of the victim agent. As we will show later in Section 4, without such guarantee, the existing attack cannot effectively explore the victim's weakness nor beating the victim. As such, unlike the state-of-the-art attack (Gleave et al., 2020), our new approach does not learn an adversarial agent by simply maximizing the rewards of the adversary through the PPO algorithm. Rather, it searches adversarial policies by providing the adversarial agent with the ability to not only maximize its rewards but, more importantly, impose negative influence upon the victim agent. For example, in the StarCraft game, this could be viewed as a learning strategy that, on the one hand, builds up a strong economy and army to countervail its opponent and, on the other hand, intervenes in the movements of the opponent and thus limits its expected rewards. Mathematically, this can be viewed as

$$J(\theta) = \text{maximize}(V_{\pi}^{\alpha}(s) - V_{\pi}^{\upsilon}(s)) \,, \qquad (1)$$

which maximizes the value function $V_{\pi}^{\alpha}(s)$ of the adversarial agent and meanwhile minimizes that of the victim agent. Here, $\pi = (\pi^{\alpha}, \pi^{\upsilon})$ is a joint policy.

While the idea mentioned above is straightforward and intuitive, using Eqn. (1) as the objective function for learning an adversarial agent still confronts two critical challenges. First, since we do not have the victim observation, we cannot directly approximate the victim value function $V_{\pi}^{\upsilon}(s)$ via a neural network that takes its observation as input. To solve this problem, we remodel this adversarial agent learning problem, transform the two-player Markov game into a one-player Markov Decision Process (MDP). As we will specify in the follow-up session, with our remodeling, the victim value function can be redefined as $V_{\pi^{\alpha}}^{\upsilon}(s)$, the output of which depends only upon the information pertaining to the adversary.

After reformalizing the victim value function, a trivial solution for resolving Eqn. (1) is to apply the vanilla policy gradient method (Konda & Tsitsiklis, 2000) and approximate both value functions with two individual neural networks (*i.e.,* $G_{\pi^{\alpha}}^{\alpha}(s)$ and $G_{\pi^{\alpha}}^{\upsilon}(s)$). However, due to the limitation rooted in the vanilla policy gradient method, such a solution cannot guarantee the monotonic changes in both value functions. As a result, a more advanced solution is to replace the $V_{\pi}^{\alpha}(s)$ in Eqn. (1) with the surrogate objective proposed in the TRPO algorithm (Schulman et al., 2015), denoted as $M_{\pi^{\alpha}}^{\alpha}(s)$. Intuitively, this surrogate objective (*i.e.,* $M_{\pi^{\alpha}}^{\alpha}(\cdot) - G_{\pi^{\alpha}}^{\upsilon}(\cdot)$) could ensure the monotonic increase of the adversarial reward. However, it cannot provide the monotonic property for the victim player. In addition, it is also unclear whether the newly added second term will break the monotonic property for the first term.[2]

Inspired by the design of TRPO, we tackle the monotonicity challenge above by designing an approximation for the expected reward difference $V_{\pi^{\alpha}}^{\alpha}(s) - V_{\pi^{\alpha}}^{\upsilon}(s)$. As we will present and prove in the follow-up section, this approximation can ensure the monotonic property for the value difference (*i.e.,* $V_{\pi_{old}^{\alpha}}^{\alpha}(s) - V_{\pi_{old}^{\alpha}}^{\upsilon}(s) \leq V_{\pi_{new}^{\alpha}}^{\alpha}(s) - V_{\pi_{new}^{\alpha}}^{\upsilon}(s)$).

3.3. Technical Details

Remodeling two-player Markov Game. A two-player Markov game (Zhang et al., 2019) can be represented as

[2]As we show in Supplementary Section S6, without ensuring the monotonic property, this more advanced method cannot train powerful adversarial agents.

$(\mathcal{N}, \mathcal{S}, \{\mathcal{A}\}^i_{i\in\mathcal{N}}, \mathcal{P}, \{R^i\}_{i\in\mathcal{N}}, \gamma)$. $\mathcal{N} = \{\alpha, \upsilon\}$ denotes the agent set, in which α and υ stand for the adversary and victim, respectively. $\mathcal{S}$ denotes the state space observed by both agents, $\mathcal{A}^i$ represents the action space of agent i. $\mathcal{P} : \mathcal{S} \times \mathcal{A}^\alpha \times \mathcal{A}^\upsilon \to \Delta(\mathcal{S})$ denotes the transition probability for any joint actions of both agents. R^i: $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function for agent i. γ is the discount factor. The state-value function for each agent is defined as a function of the joint policy $\pi(a|s) = \prod_{i\in\mathcal{N}} \pi^i(a^i|s)$

$$V^i_\pi(s) = \mathbb{E}_{a\sim\pi(a|s)}[R^i(s,a) + \gamma\mathbb{E}_{s'\sim\mathcal{P}}[V^i_\pi(s')]]\,. \tag{2}$$

As is mentioned in Section 3.1, the victim agent follows a fixed policy. Under this setup, we have the following proposition (see the proof in Supplementary Section S1).

Proposition 1. *In a two-player Markov game, if one agent follows a fixed policy, the state transition of the game system will depend only upon the policy of the other agent rather than their joint policy.*

With the proposition above, we can redefine both agents' state-value and action-value functions as the functions of the adversarial policy below.

$$\begin{aligned} &Q^i_{\pi^\alpha}(s,a^\alpha) = R^i(s,a) + \mathbb{E}_{s'\sim P(s'|s,a^\alpha)}[\gamma V^i_{\pi^\alpha}(s')]\,, \\ &V^i_{\pi^\alpha}(s) = \mathbb{E}_{a^\alpha\sim\pi^\alpha}[Q^i_{\pi^\alpha}(s,a^\alpha)]\,, \end{aligned} \tag{3}$$

where $a = (a^\alpha, F^{\pi^\upsilon}(s))$. $F^{\pi^\upsilon}(s)$ represent the actions sampled from the fixed victim policy.

As is shown above, the new state-value and action-value functions no longer enclose the policy of the victim nor its observations or actions. It perfectly addresses the concern of having to know the victim agent.

Constructing and Solving Objective Function. With the new definition above, the objective function shown in Eqn. (1) can be reformalized as

$$J(\theta) = \text{maximize}_\theta(V^\alpha_{\pi^\alpha_\theta}(s) - V^\upsilon_{\pi^\alpha_\theta}(s))\,. \tag{4}$$

Here, π^α_θ refers to as the adversarial policy network parameterized by θ. As is discussed in Section 3.2, trivial extending neither the vanilla policy gradient method nor the TRPO algorithm could guarantee the monotonic property. To address this problem, we propose the following method.

Based on the Theorem 1 in Schulman et al. (2015), we can approximate $V^\alpha_{\pi^\alpha}(s)$ with $M^\alpha_{\pi^\alpha}(\pi^{\alpha'})$ and thus obtain the following relationship

$$\begin{aligned} &V^\alpha_{\pi^{\alpha'}}(s) \geq M^\alpha_{\pi^\alpha}(\pi^{\alpha'})\,, \\ &M^\alpha_{\pi^\alpha}(\pi^{\alpha'}) = L^\alpha_{\pi^\alpha}(\pi^{\alpha'}) - C_1\mathbb{KL}^{\max}(\pi^\alpha(\cdot|s)||\pi^{\alpha'}(\cdot|s))\,, \end{aligned} \tag{5}$$

where C_1 is a constant. $\mathbb{KL}^{\max}(\pi^\alpha(\cdot|s)||\pi^{\alpha'}(\cdot|s)) = \max_s\mathbb{KL}(\pi^\alpha(\cdot|s)||\pi^{\alpha'}(\cdot|s))$. $\pi^{\alpha'}$ and π^α refers to as the new and old adversarial policy, respectively.

To obtain an approximation for $V^\upsilon_{\pi^\alpha}(s)$, we follow the idea in Schulman et al. (2015) and propose the method below. First, we compute the difference of the victim's state-value function before and after the update of the adversarial policy and come up with the following lemma (see the proof in Supplementary Section S2).

Lemma 1. *Given an old policy π^α and a new policy $\pi^{\alpha'}$ in a two-agent Markov game, the difference of the victim state-value function under each policy is as follows.*

$$V^\upsilon_{\pi^{\alpha'}}(s) - V^\upsilon_{\pi^\alpha}(s) = E_{\tau\sim\pi^{\alpha'}}[\sum_{t=0}^{\infty}\gamma^t A^\upsilon_{\pi^\alpha}(s_t, a^\alpha_t)]\,. \tag{6}$$

Intuitively, Lemma 1 indicates that, if the adversarial agent switches to a new policy $\pi^{\alpha'}$ from the old one π^α at the state s, the expectation of the victim's future reward will change. Technically, one can compute the change by applying the victim's advantage function under the old adversarial policy to the trajectories collected by using the new adversarial policy.

To get rid of the summation over infinite time in Eqn. (6), we can further rewrite this equation by taking the summation over all possible states.

$$V^\upsilon_{\pi^{\alpha'}}(s) - V^\upsilon_{\pi^\alpha}(s) = \sum_s \rho_{\pi^{\alpha'}}(s)\sum_a \pi^{\alpha'}(a^\alpha|s)A^\upsilon_{\pi^\alpha}(s,a^\alpha), \tag{7}$$

where $\rho_{\pi^{\alpha'}}(s) = \sum_t \gamma^t P(s_t = s|\pi^{\alpha'})$.

With the rewritten equation in hand, the computation of Eqn. (7) is still hard due to the unknown visitation frequencies of $\pi^{\alpha'}$. To solve this problem, we replace $\rho_{\pi^{\alpha'}}(s)$ with $\rho_{\pi^\alpha}(s)$ and thus approximate $V^\upsilon_{\pi^{\alpha'}}(s)$ with $L^\upsilon_{\pi^\alpha}(\pi^{\alpha'}) = V^\upsilon_{\pi^\alpha}(s) + \sum_s \rho_{\pi^\alpha}(s)\sum_a \pi^{\alpha'}(a^\alpha|s)A^\upsilon_{\pi^\alpha}(s,a^\alpha)$. The relationship between our approximation and the actual value function is described in the theorem below (see the proof in Supplementary Section S3).

Theorem 1. *The difference between $V^\upsilon_{\pi^{\alpha'}}(s)$ and $L^\upsilon_{\pi^\alpha}(\pi^{\alpha'})$ is bounded by:*

$$\begin{aligned} &V^\upsilon_{\pi^{\alpha'}}(s) \leq L^\upsilon_{\pi^\alpha}(\pi^{\alpha'}) + C_2\mathbb{KL}^{\max}(\pi^\alpha||\pi^{\alpha'}) = M^\upsilon_{\pi^\alpha}(\pi^{\alpha'})\,, \\ &C_2 = \frac{4\max_{s,a^\alpha}|A^\upsilon_{\pi^\alpha}(s,a^\alpha)|\gamma}{(1-\gamma)^2}\,. \end{aligned} \tag{8}$$

Using the inequality in the theorem above along with that in Eqn. (5), we can easily derive the following inequality

$$V^\alpha_{\pi^{\alpha'}}(s) - V^\upsilon_{\pi^{\alpha'}}(s) \geq M^\alpha_{\pi^\alpha}(\pi^{\alpha'}) - M^\upsilon_{\pi^\alpha}(\pi^{\alpha'}) = M_{\pi^\alpha}(\pi^{\alpha'})\,. \tag{9}$$

Further, we can derive

$$M_{\pi^\alpha}(\pi^\alpha) = L^\alpha_{\pi^\alpha}(\pi^\alpha) - L^\upsilon_{\pi^\alpha}(\pi^\alpha) = V^\alpha_{\pi^\alpha} - V^\upsilon_{\pi^\alpha}\,. \tag{10}$$

Then, by maximizing $M_{\pi^\alpha}(\pi^{\alpha'})$ via gradient ascent at each iteration, we can ensure $M_{\pi^\alpha}(\pi^\alpha) \leq M_{\pi^\alpha}(\pi^{\alpha'})$

and thus guarantee the monotonic property for Eqn. (4) (*i.e.,* $(V^{\alpha}_{\pi^{\alpha}}(s) - V^{\upsilon}_{\pi^{\alpha}}(s)) \leq (V^{\alpha}_{\pi^{\alpha'}}(s) - V^{\upsilon}_{\pi^{\alpha'}}(s))$). Specifically, $M_{\pi^{\alpha}}(\pi^{\alpha'})$ equals to the following Equation

$$M_{\pi^{\alpha}}(\pi^{\alpha'}) = \sum_{s} \rho_{\pi^{\alpha}}(s) \sum_{a} \pi^{\alpha'}(a^{\alpha}|s)(A^{\alpha}_{\pi^{\alpha}}(s, a^{\alpha}) - A^{\upsilon}_{\pi^{\alpha}}(s, a^{\alpha})) - C\mathbb{KL}^{\max}(\pi^{\alpha}||\pi^{\alpha'}) + C_3 \,, \tag{11}$$

where $C = C_1 - C_2$ and $C_3 = (V^{\alpha}_{\pi^{\alpha}}(s) - V^{\upsilon}_{\pi^{\alpha}}(s))$ are constants. To solve this Eqn. (11), we first follow the TRPO algorithm and transform the $C\mathbb{KL}^{\max}(\pi^{\alpha}||\pi^{\alpha'})$ into a trust region constraint over the average KL-divergence. Even after transformation, the optimization is still hard to implement due to the summation over the new policy $\pi^{\alpha'}$. To solve this problem, we apply importance sampling and replace it with the summation over the old policy that can be computed via the Monte Carlo method (Thrun, 2000). Then, we further replace the trust region constrains with clipped ratio operation introduced in the PPO algorithm and obtain our final optimization function.

$$\begin{aligned} &\text{argmax}_{\theta}\ \mathbb{E}_{(a^{\alpha}_t, s_t) \sim \pi^{\alpha}_{old}}[\min(\text{clip}(\rho_t, 1-\epsilon, 1+\epsilon)A^{\alpha}_t, \rho_t A^{\alpha}_t) \\ &- \min(\text{clip}(\rho_t, 1-\epsilon, 1+\epsilon)A^{\upsilon}_t, \rho_t A^{\upsilon}_t)] \,, \\ &\rho_t = \frac{\pi^{\alpha}_{\theta}(a^{\alpha}_t|s_t)}{\pi^{\alpha}_{old}(a^{\alpha}_t|s_t)}, A^{\alpha}_t = A^{\alpha}_{\pi^{\alpha}_{old}}(a^{\alpha}_t, s_t), A^{\upsilon}_t = A^{\upsilon}_{\pi^{\alpha}_{old}}(a^{\alpha}_t, s_t) \,. \end{aligned} \tag{12}$$

To resolve an adversarial policy, we first approximate the adversarial and victim value function with two neural networks through the TD-Learning (Tesauro, 1995) and then update the adversarial policy network by optimizing Eqn. (12). For a more detailed description of how to obtain Eqn. (12) and our learning algorithm, readers could refer to the Supplementary Section S4.

4. Evaluation

In this section, we evaluate our proposed learning algorithm by using five selected games (*i.e.,* four MuJoCo games and StarCraft II). Due to space limit, we specify the implementation details and experiment setup (*i.e.,* game and victim policy selection, evaluation metric, hyperparameters) in Supplementary Section S5.

4.1. Exploitability

Experiment Design. In our first experiment, we use the state-of-the-art approach (Gleave et al., 2020) as our baseline and compare it with our proposed attack. To be specific, given a two-player game as well as a victim agent well-trained for that game, we train our adversarial agent against the victim by using the proposed learning algorithm and compare its winning rate with that collected from the state-of-the-art method.

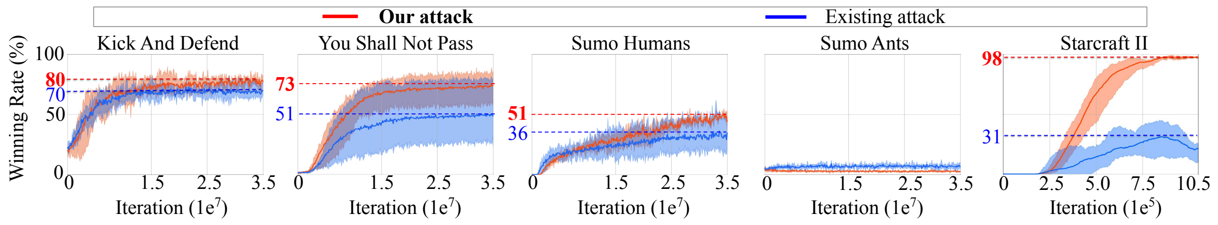
Experiment Results. Figure 1 shows the winning rate comparison between our adversarial agents and that obtained by the existing attack. As we can observe first, overall, the adversarial agent trained by our proposed method demonstrates higher winning rates (or in other words, stronger exploitability) than that prepared by the state-of-the-art technique (Gleave et al., 2020). This result confirms that by maximizing the total reward difference between the adversary and victim, our method possesses more potential to discover an effective adversarial policy and thus demonstrates higher winning rates than the existing attack that only maximizes the adversarial expected total reward.

Figure 1(a) also shows that our method demonstrates a more significant increase in the winning rate on StarCraft II game (98% vs. 31%) than MuJoCo games. It is because for MuJoCo games, the increase in the adversarial agent's reward, to some extent, contributes to the decrease in the victim's reward gain. However, the StarCraft II has a more sophisticated intermediate reward system, in which the increase in adversarial reward has minimal impact on the victim reward gain.

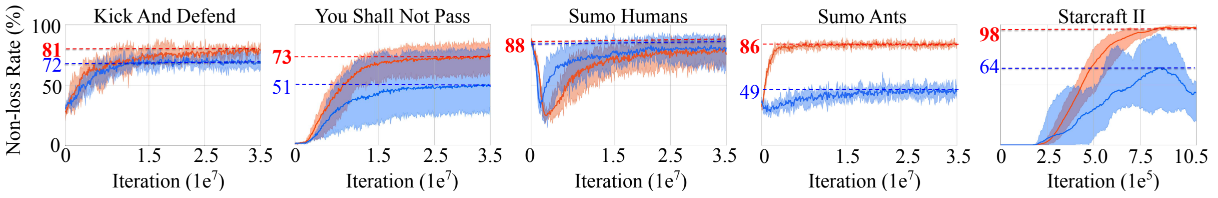
In addition to the adversarial winning rate, we also compare the behaviors of the adversarial agents learned through our proposed method and that learned through the baseline approach. We illustrate the agent behaviors through demo videos at `https://tinyurl.com/y3ax4ayk`. On the three MuJoCo games (*i.e.,* You-Shall-Not-Pass, Kick-And-Defend, and Sumo-Humans), similar to the agent learned through the baseline, our adversarial agent also establish weird behaviors and trick the victim into behaving in an undesired fashion. While the abnormal behaviors are similar for these three games, the adversarial agent learned through our method demonstrates stronger exploitability (See Figure 1(a)).

As we also observe from Fig. 1(a), both the baseline and our method fail to identify an effective policy to master the Sumo-Ants game. As is discussed in Gleave et al. (2020), this is because the observation space under the control of an adversary is low, leaving less room for it to exploit the weakness of the victim via its actions. However, we argue, even if we observe the similar winning rate in existing and our approaches, this does not imply our proposed method is as futile as the existing approach. In Figure 1(b), for each game, we show the combination of winning and tie rates (*i.e.,* non-loss rate). As we can observe, for Sumo-Ants, our method could still obtain an adversarial agent that significantly prevents the victim from receiving sufficient wins (*i.e.,* about 88% winning plus tie rates for the adversary).

Following the adversarial agent's exploitability comparison, we also compare the victim agent's behaviors when it confronts two different adversarial agents in this game. As we can observe from the demo videos (`https://tinyurl.com/yxteeyuo`), our adversarial agent



(a) The winning rates of the adversarial agents.



(b) The non-loss (*i.e.,* winning plus tie) rates of the adversarial agents.

Figure 1. The performance comparison of adversarial agents against the victim. The adversarial agents obtained from two different approaches – our proposed method and an existing method (Gleave et al., 2020). Note that the darker solid lines represent the average winning (plus tie) rates, whereas the lighter bands indicate the corresponding variations between the maximal and minimal winning (plus tie) rates. The highlighted y-axis labels are the highest average winning/non-loss rates over the total amount of iterations. In addition to visualizing the winning rate, Supplement Section S6 also conducts a statistical test to further confirm the significance of our method. Note that Fig. S3 in the Supplement further shows the changes in victim agents' winning rate during the attack training.

learns to intentionally jump onto the ground, while the agent learned through the baseline approach keeps still. To understand this behavior difference, we take a closer look at the game rule of the Sumo-Ants game and discover an unfairness design of the game. As we detail in Supplementary Section S5, if any player falls from the arena without contacting its opponent, the game will count the match as a draw. In our experiment, our adversarial agent learns to exploit this rule, preventing the victim agent from winning the game. This result indicates that our attack could find and abuse the game unfairness when the adversary has minimal impact on the victim's observation space. Unlike our attack, the baseline does not demonstrate the capability of exploiting the game's unfairness and thus fail to establish sufficient exploitability against the victim.

Finally, the light color bands in Fig. 1 shows the performance variance of each method. As we can observe that our proposed method generally has less variation in agent performance than the existing technique. It means that our proposed algorithm is more robust against those randomness factors, such as the initial game state and the probabilistic state transition. By following our algorithm, an attacker could, therefore, learn an adversarial agent with consistent performance even if the game initializes his agent at different random places in the adversarial learning process.

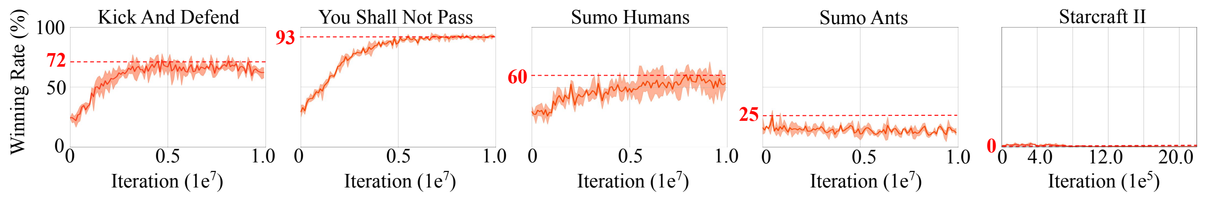
In this work, we also conduct some other comparison experiments. Due to space limit, we present them in the Supplementary Section S6. For example, we compare our method with a method which utilizes only the second term in Eqn. (12) as the objective function (*i.e.,* $\text{argmax}_\theta - \mathbb{E}[\min(\text{clip}(\rho_t, 1-\epsilon, 1+\epsilon)A_t^v, \rho_t A_t^v)]$). This comparison helps assess whether focusing only on reducing the victim's reward can lead to the same attack effect. Besides, we also compare our method with an approach without the monotonic property. This comparison helps us double confirm the importance of monotonicity.

Table 1. The performance of the victims against corresponding regular agents before and after the adversarial retraining. The numbers on the gray canvas represent the winning rates, whereas those on the white one indicate the win plus tie rate.

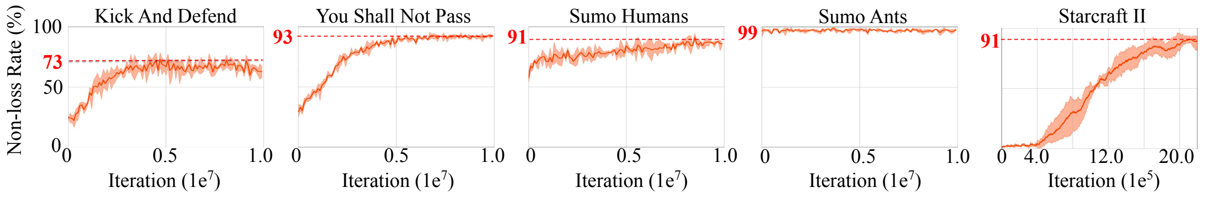
Game	After retraining (%)		Before retraining (%)	
Kick And Defend	12.0	15.0	12.0	14.0
You Shall Not Pass	59.0	59.0	60.0	60.0
Sumo Humans	81.0	92.0	75.0	87.0
Sumo Ants	41.0	59.0	34.0	55.0
StarCraft II	83.0	87.0	46.0	47.0

4.2. Adversary Resistance

Experiment Design. Using the method training the adversary, Gleave et al. (2020) demonstrate that one could retrain the victim and thus improve its adversary resistance. In this experiment, we follow their experimental setup, using the way we train the adversary to retrain the corresponding victim agent. Then, we examine the adversary resistance of the retrained agent. Further, we explore the generalizability of the retrained victim agents. Specifically, we set the retrained victim agent to play with other regular agents for



(a) The winning rates of the adversarial-retrained victim agent against our adversarial agent in five games.



(b) The non-loss rates of the adversarial-retrained victim agent against our adversarial agent in five games.

Figure 2. The performance of victim agents after being retrained by using our proposed learning method.

100 rounds and report its winning rate. Through this setup, we study whether the adversarial training wipes out the retrained agent's ability to play with other regular agents.

Last but not least, we also measure the retrained victim agent's robustness against our attack and its robustness against the baseline attack (Gleave et al., 2020). Specifically, we take the victim agent retrained against our adversarial attack to play with the adversarial agent learned through the baseline and vice versa. In this process, we record the winning rates of the retrained victim agent and examine whether adversarial training against our proposed attack could lead to a more adversary-resistance agent.

Experiment Results. In Figure 2, we show the performance of the victim agent after being retrained.[3] We can observe from the figure that, for all of the five games, the adversarial training takes effect, helping the victim agent pick up the ability to minimize the influence of adversarial agents upon itself. For example, the winning rate of the retrained agent returns 96% in the "You-Shall-Not-Pass" game. For others, the win plus tie rates bounce back to $> 70\%$. As is also shown in table 1, adversarial retraining imposes a negligible influence on the victim agent's ability to play with another regular agent. It indicates that by retraining the victim agent with a mix of adversarial episodes and normal episodes, the retrained agent could establish the adversarial robustness while preserving its generalizability against regular agents. In Supplementary Section S6, we study the influence of the episode split upon the robustness and generalizability of the retrained victim agent.

Table 2 shows the performance of retrained agents against different adversarial agents. As we can first observe from the table that the victim agent retrained against our attack is capable of defeating the adversarial agent obtained by the state-of-art attack (*i.e.,* Column 3). It could achieve the similar robustness to the victim retrained against the baseline (*i.e.,* Column 3 vs. 5). On the contrary, when playing with our adversarial agent, the baseline retrained victim is less effective (*i.e.,* Column 4). It cannot achieve comparable performance with our retrained victim (*i.e.,* Column 1 vs. 4). The result indicates that an attack with a strong exploitability could help the victim agent pick up a robust policy, which is also resistant to an adversarial policy with a weaker exploitability. This finding suggests that users should utilize the more powerful adversarial agent for adversarial retraining and expect the retrained agent could defeat weaker adversaries.

Table 2. The robustness comparison of the victim retrained against our attack and retrained against the baseline attack (Gleave et al., 2020). The numbers in the table are the win plus tie of the retrained victim agent. "Base" and "adv." refers to "baseline" and "adversary", respectively.

Games	Our retrained victim (%)		Baseline retrained victim (%)	
	vs. Our adv.	vs. Base adv.	vs. Our adv.	vs. Base adv.
Kick And Defend	68.0	66.0	35.0	73.0
You Shall Not Pass	93.0	94.0	86.0	91.0
Sumo Humans	90.0	82.0	71.0	80.0
Sumo Ants	98.0	91.0	91.0	93.0
StarCraft II	89.0	99.0	4.0	87.0

4.3. Root Cause Analysis

Experiment Design. To further understand the root cause behind adversarial policy's effectiveness, we follow Gleave et al. (2020) and conduct the following experiment. First, we blind the victim's observation on the adversary or, in other words, zero out the victim observation pertaining to the adversary. Then, we test the partially blind vic-

[3]Table S1 in Supplement shows the victim agents' performances against our adversarial agents before retraining.

Table 3. The win plus tie rate of the (blind) victim agent against our adversarial agent and that obtained by the baseline attack (Gleave et al., 2020). "Before" and "After" indicates before and after blinding the victim observations, respectively.

Games	Our attack (%)		Baseline attack (%)	
	Before	After	Before	After
Kick And Defend	14.0	94.0	45.0	97.0
You Shall Not Pass	26.0	98.0	48.0	97.0
Sumo Humans	53.0	67.0	65.0	68.0
Sumo Ants	86.0	87.0	91.0	95.0
StarCraft II	2.0	24.0	64.0	98.0

tim against our adversarial agent and that learned from the state-of-the-art method Gleave et al. (2020). Using this experiment, we demonstrate that adversarial agent not only can win by taking actions to induce natural observations that are adversarial to the victim but also can win by exploiting game unfairness.[4]

Experiment Results. Table 3 shows the victim's win plus tie rate before and after blinding. Similar to the findings in Gleave et al. (2020), the victim winning rates against both attacks increase dramatically in the games – You-Shall-Not-Pass and Kick-And-Defend. This result aligns the results present in Gleave et al. (2020). It indicates that, for these two games, the adversarial agent wins the victim by indirectly influencing the victim's observation. However, we also discover the blinding has no impact on our attack in the Sumo-Ants game. This finding confirms that, for this game, our adversarial agent beats the victim by exploiting the game unfairness. Last but not least, our attack establishes a lower adversarial winning rate drop than the existing attack on the Starcraft II games. This confirms that our attack has a stronger exploitability than the existing attack on this sophisticated game.

5. Discussion and Future Work

Transferability. In Supplementary Section S7, we show that both our attack and the state-of-art attack establish a certain level of transferability. Specifically, we find that adversarial policies that explore game unfairness have stronger transferability than those disturbing the victim observation through weird actions. The above observations are obtained when varying only hyperparameters and initial states. Recent works (Huang et al., 2017) on the transferability of the observation manipulation attacks show that the transferability also relies upon many other factors, such as games themselves and learning algorithms. In the future, we will thoroughly evaluate the transferability of our proposed attack under different setups.

Other games. In addition to the two-player Markov game studied in this work, there are two other types of popular competitive games – two-player extensive-form games and two-player multi-agent games (Zhang et al., 2019).[5] Regarding the first type, at any given time step, only one agent can observe the game state and thus take action. As such, the state transition of this type of game depends solely upon the action of one agent. Existing research (Srinivasan et al., 2018) has extended the vanilla policy gradient to train a DRL agent in extensive-form games, such as Poker (Lanctot et al., 2019). Based on the learning method proposed in (Srinivasan et al., 2018), our attack can be potentially generalized to this game. The key challenge is how to enable monotonicity for the adversarial learning algorithm. In the future, we will explore how to design the learning objective function to guarantee monotonicity. Concerning the two-player multi-agent games, the agents in one team cooperate with each other to compete against the agents from the other team. Enabling an adversarial attack in this game is equivalent to training a set of cooperative agents to defeat another set of well-trained agents. As discussed in (Daskalakis et al., 2009), the key challenge of preparing a group of cooperative agents is how to handle the information sharing among the agents. Recent research (Zhang et al., 2018) has explored how to extend the vanilla policy gradient to a multi-agent setting. As part of future work, we will borrow the idea from this technique and generalize our attack to multi-agent settings.

Zero-sum game vs. two-player game. As is mentioned in Section 3.1, in most real-world two-player games, the reward design of the game usually does not follow the rule of zero-sum.[6] As such, our proposed method could demonstrate a performance advantage over the state-of-the-art method (Gleave et al., 2020) (which is built on top of the PPO algorithm). However, we admit that if a two-player game follows the zero-sum rule restrictively, in theory, the adversarial agent trained by our approach will demonstrate the same performance as that prepared by Gleave et al. (2020). In other words, under the zero-sum setup, our objective function could be viewed as maximizing the adversarial expected reward, which is equivalent to the objective function of the PPO method (See Supplement S8 for the demonstration of the equality of our attack and Gleave et al. (2020) in a zero-sum game). While this could potentially become a limitation of our work, we argue this does

[4]In Supplement S6, we also follow Gleave et al. (2020) and conduct an in-depth analysis on the activations gathered from the victim policy networks. The analysis further demonstrates the behavior differences between our adversarial agent and that learned from the attack in Gleave et al. (2020).

[5]Existing research (Daskalakis et al., 2009) has proven that training an RL agent for multi-player games is an NP problem. As such, we do not consider multi-player games in this work.

[6]Such games can also be taken as general-sum competitive games.

not dilute our contribution. As is discussed above, to obtain optimal performance, most two-player games, if not all, do not follow the zero-sum rule strictively. Therefore, it leaves room for an attacker to leverage our approach for exploiting the weakness of agents in the game.

6. Conclusion

In two-player Markov games, existing methods for learning an adversarial policy either make unrealistic assumptions or fail to demonstrate sufficient advantages over target agents (*i.e.,* victim agents), especially when the game does not strictly follow the zero-sum setup. In this work, we develop a new attack against a victim agent trained by DRL in the context of two-player games. Technically, we carefully design the objective function of our adversarial learning algorithm such that the agent trained by our attack could guarantee to increase the expected reward difference between the adversary and victim monotonically. By using five games commonly utilized in reinforcement learning evaluation, we show that our attack not only significantly outperforms the state-of-the-art attack, but also demonstrates an ability to abuse the unfairness of the target game. With this discovery, we safely conclude that our newly proposed attack could help an attacker learn an adversarial agent much more effective in defeating victims.

Acknowledgments

We would like to thank the anonymous reviewers and Meta reviewer for their helpful comments. This project was supported in part by NSF grant CNS-1718459 and CNS-1954466, by ONR grant N00014-20-1-2008, by the Amazon Research Award, and by the IBM Ph.D. Fellowship Award.

References

ATARI. Atari games. https://www.atari.com/, 2006.

Bansal, T., Pachocki, J., Sidor, S., Sutskever, I., and Mordatch, I. Emergent complexity via multi-agent competition. In *Proc. of ICLR*, 2018.

Behzadan, V. and Munir, A. Vulnerability of deep reinforcement learning to policy induction attacks. In *Proc. of MLDM*, 2017.

Carlini, N. and Wagner, D. Towards evaluating the robustness of neural networks. In *Proc. of S&P*, 2017.

Daskalakis, C., Goldberg, P. W., et al. The complexity of computing a nash equilibrium. *SIAM Journal on Computing*, 2009.

Gleave, A., Dennis, M., Kant, N., Wild, C., Levine, S., and Russell, S. Adversarial policies: Attacking deep reinforcement learning. In *Proc. of ICLR*, 2020.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. In *Proc. of ICLR*, 2015.

Huang, S., Papernot, N., Goodfellow, I., Duan, Y., and Abbeel, P. Adversarial attacks on neural network policies. In *Proc. of ICLR workshop*, 2017.

Kiourti, P., Wardega, K., Jha, S., and Li, W. Trojdrl: Trojan attacks on deep reinforcement learning agents. *arXiv preprint arXiv:1903.06638*, 2019.

Konda, V. R. and Tsitsiklis, J. N. Actor-critic algorithms. In *Proc. of NeurIPS*, 2000.

Kos, J. and Song, D. Delving into adversarial attacks on deep policies. In *Proc. of ICLR Workshop*, 2017.

Lanctot, M., Lockhart, E., Lespiau, J.-B., Zambaldi, V., Upadhyay, S., Pérolat, J., Srinivasan, S., Timbers, F., Tuyls, K., Omidshafiei, S., et al. Openspiel: A framework for reinforcement learning in games. *arXiv preprint arXiv:1908.09453*, 2019.

Lee, X. Y., Ghadai, S., Tan, K. L., Hegde, C., and Sarkar, S. Spatiotemporally constrained action space attacks on deep reinforcement learning agents. In *Proc. of AAAI*, 2020.

Lin, J., Dzeparoska, K., Zhang, S. Q., Leon-Garcia, A., and Papernot, N. On the robustness of cooperative multi-agent reinforcement learning. In *Proc. of DLS Workshop*, 2020.

Lin, Y.-C., Hong, Z.-W., Liao, Y.-H., Shih, M.-L., Liu, M.-Y., and Sun, M. Tactics of adversarial attack on deep reinforcement learning agents. In *Proc. of IJCAI*, 2017.

Lykouris, T., Simchowitz, M., Slivkins, A., and Sun, W. Corruption robust exploration in episodic reinforcement learning. *arXiv preprint arXiv:1911.08689*, 2019.

Ma, Y., Zhang, X., Sun, W., and Zhu, J. Policy poisoning in batch reinforcement learning and control. In *Proc. of NeurIPS*, 2019.

Madry, A., Makelov, A., Schmidt, L., et al. Towards deep learning models resistant to adversarial attacks. In *Proc. of ICLR*, 2018.

OpenAI. Roboschool: open-source software for robot simulation. https://openai.com/blog/roboschool/, 2017.

Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., and Swami, A. The limitations of deep learning in adversarial settings. In *Proc. of EuroS&P*, 2016.

Russo, A. and Proutiere, A. Optimal attacks on reinforcement learning policies. *arXiv preprint arXiv:1907.13548*, 2019.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *Proc. of ICML*, 2015.

Srinivasan, S., Lanctot, M., Zambaldi, V., Pérolat, J., Tuyls, K., Munos, R., and Bowling, M. Actor-critic policy optimization in partially observable multiagent environments. In *Proc. of NeurIPS*, 2018.

Sun, J., Zhang, T., Xie, X., Ma, L., Zheng, Y., Chen, K., and Liu, Y. Stealthy and efficient adversarial attacks against deep reinforcement learning. In *Proc. of AAAI*, 2020.

Sun, P., Sun, X., Han, L., Xiong, J., Wang, Q., Li, B., Zheng, Y., Liu, J., Liu, Y., Liu, H., et al. Tstarbots: Defeating the cheating level builtin ai in starcraft ii in the full game. *arXiv preprint arXiv:1809.07193*, 2018.

Tesauro, G. Temporal difference learning and td-gammon. *Communications of The ACM*, 1995.

Thrun, S. Monte carlo pomdps. In *Proc. of NeurIPS*, 2000.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *Proc. of ICIRS*, 2012.

Unity. Unity machine learning agents toolkit. https://unity3d.com/machine-learning/, 2020.

Xiao, C., Pan, X., He, W., Peng, J., Sun, M., Yi, J., Li, B., and Song, D. Characterizing attacks on deep reinforcement learning. *arXiv preprint arXiv:1907.09470*, 2019.

Yang, Z., Iyer, N., et al. Design of intentional backdoors in sequential models. *arXiv preprint arXiv:1902.09972*, 2019.

Zhang, H., Chen, H., Boning, D., and Hsieh, C.-J. Robust reinforcement learning on state observations with learned optimal adversary. In *Proc. of ICLR*, 2021.

Zhang, K., Yang, Z., Liu, H., Zhang, T., and Başar, T. Fully decentralized multi-agent reinforcement learning with networked agents. *arXiv preprint arXiv:1802.08757*, 2018.

Zhang, K., Yang, Z., and Başar, T. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *arXiv preprint arXiv:1911.10635*, 2019.

Zhao, Y., Shumailov, I., Cui, H., Gao, X., Mullins, R., and Anderson, R. Blackbox attacks on reinforcement learning agents using approximated temporal information. *arXiv preprint arXiv:1909.02918*, 2019.

Adversarial Policy Learning in Two-player Competitive Games

S1 Proof of Proposition 1

Proposition 1. *In a two-player Markov game, if one agent follows a fixed policy, the state transition of the game system will depend only upon the policy of the other agent rather than their joint policies.*

Proof. Without the loss of generalizability, we denote the agent as α and υ, and assume the agent υ follows a fixed policy. At state s_t, the probability of taking the joint actions $(a_t^\alpha, a_t^\upsilon)$ and transiting to s_{t+1} is

$$\begin{aligned} P(s_{t+1}, a_t^\alpha, a_t^\upsilon | s_t) =& P(s_{t+1}|a_t^\alpha, a_t^\upsilon, s_t) P(a_t^\alpha, a_t^\upsilon | s_t) = P(s_{t+1}|a_t^\alpha, a_t^\upsilon, s_t) P(a_t^\alpha | a_t^\upsilon, s_t) \pi^\upsilon(a_t^\upsilon|s_t) \\ =& c \cdot P(s_{t+1}|a_t^\alpha, a_t^\upsilon, s_t) P(a_t^\alpha|a_t^\upsilon, s_t) = c \cdot P(s_{t+1}|a_t^\alpha, a_t^\upsilon, s_t) \pi^\alpha(a_t^\alpha|s_t)\,, \end{aligned} \tag{1}$$

where $P(a_t^\upsilon|s_t) = c$. Given that at a time step t, an action of the agent a_t^α depends only upon the current state s_t, we have $\pi^\alpha(a_t^\alpha|s_t) = P(a_t^\alpha|a_t^\upsilon, s_t)$.

As we can observe from Eqn. (1), during the adversarial training process, the only changed part is policy π^α. As such, the change in π^α determines the change in state transition and the in change both agent's value functions. To be specific, given a set of trajectories $\{\tau_1, \ldots, \tau_M\}$, the state-value function of α agent V_π^α can be computed by

$$V_\pi^\alpha = \sum_{m=1}^{M} R^\alpha(\tau_m) P(\tau_m; \theta)\,. \tag{2}$$

The state-value function of υ agent V_π^υ can be computed by

$$V_\pi^\upsilon = \sum_{m=1}^{M} R^\upsilon(\tau_m) P(\tau_m; \theta)\,. \tag{3}$$

In Eqn. (2) and Eqn. (3),

$$\begin{aligned} P(\tau;\theta) &= P(s_0) \prod_{t=1}^{T} P(s_t, a_{t-1}^\alpha, a_{t-1}^\upsilon | s_{t-1}) \\ &= P(s_0) \prod_{t=1}^{T} P(s_t, a_{t-1}^\alpha | a_{t-1}^\upsilon, s_{t-1}) P(a_{t-1}^\upsilon | s_{t-1})\,. \end{aligned} \tag{4}$$

Since the victim agent follows a fixed policy, $P(a_{t-1}^\upsilon|s_{t-1}) = c$. Then, Eqn. (4) can be rewrote as

$$\begin{aligned} P(\tau;\theta) &= P(s_0) \prod_{t=1}^{T} P(s_t, a_{t-1}^\alpha, a_{t-1}^\upsilon | s_{t-1}) \\ &= P(s_0) \prod_{t=1}^{T} c \cdot P(s_t | a_{t-1}^\alpha, a_{t-1}^\upsilon, s_{t-1}) \pi^\alpha(a_{t-1}^\alpha | s_{t-1})\,. \end{aligned} \tag{5}$$

Similar to Eqn. (1), π^α is the only changed component in Eqn. (5). Plugging Eqn. (5) into either Eqn. (2) or Eqn. (3), we can find out the change in both agent's state-value functions depend only upon the policy π^α. Combining these observations with the aforementioned observation in Eqn. (1), we can conclude that the change in π^α determines the change in state transition as well as the change in both agent's value functions. □

S2 Proof of Lemma 1

Lemma 1. *Given a old policy π^α and a new policy $\pi^{\alpha'}$ in a two-agent Markov game, the difference of the victim state-value function under each policy is as follows.*

$$V^{\upsilon}_{\pi^{\alpha'}}(s) - V^{\upsilon}_{\pi^\alpha}(s) = E_{\tau\sim\pi^{\alpha'}}[\sum_{t=0}^{\infty}\gamma^t A^{\upsilon}_{\pi^\alpha}(s_t, a^\alpha_t)]\,. \tag{6}$$

Proof. Recall that in Section 3.3, we state that the victim value function can be redefined as follows

$$Q^{\upsilon}_{\pi^\alpha}(s_t, a^\alpha_t) = R^\upsilon(s_t, a_t) + \gamma E_{s_{t+1}|s_t,a_t}[V^{\upsilon}_{\pi^\alpha}(s_{t+1})]\,. \tag{7}$$

In addition, the tower property of conditional expectations gives the following equation

$$E_{X,Y}[f(x,y)] = E_X E_{Y|X}[f(x,y)] = E_{X,Y}E_{Y|X}[f(x,y)]\,, \tag{8}$$

where x and y are random variables. Based on Eqn. (8), we also have

$$E_{\tau\sim\pi^{\alpha'}}V^{\upsilon}_{\pi^\alpha}(s_{t+1}) = E_{\tau\sim\pi^{\alpha'}}[E_{s_{t+1}|s_t,a_t}[V^{\upsilon}_{\pi^\alpha}(s_{t+1})]]\,. \tag{9}$$

Then, we can compute the victim state-value function difference.

$$\begin{aligned}
V^{\upsilon}_{\pi^{\alpha'}}(s) - V^{\upsilon}_{\pi^\alpha}(s) =& E_{\tau\sim\pi^{\alpha'}}[\sum_{t=0}^{\infty}\gamma^t R^\upsilon(s_t,a_t)] - V^{\upsilon}_{\pi^\alpha}(s) \\
=& E_{\tau\sim\pi^{\alpha'}}[\sum_{t=0}^{\infty}\gamma^t[R^\upsilon(s_t,a_t) - V^{\upsilon}_{\pi^\alpha}(s) + V^{\upsilon}_{\pi^\alpha}(s)]] - V^{\upsilon}_{\pi^\alpha}(s) \\
=& E_{\tau\sim\pi^{\alpha'}}[\sum_{t=0}^{\infty}\gamma^t R^\upsilon(s_t,a_t) - \sum_{t=0}^{\infty}\gamma^t V^{\upsilon}_{\pi^\alpha}(s) + \sum_{t=0}^{\infty}\gamma^t V^{\upsilon}_{\pi^\alpha}(s)] - V^{\upsilon}_{\pi^\alpha}(s) \\
=& E_{\tau\sim\pi^{\alpha'}}[\sum_{t=0}^{\infty}\gamma^t[R^\upsilon(s_t,a_t) - V^{\upsilon}_{\pi^\alpha}(s)] + \sum_{t=0}^{\infty}\gamma^{t+1} V^{\upsilon}_{\pi^\alpha}(s_{t+1}) + V^{\upsilon}_{\pi^\alpha}(s)] - V^{\upsilon}_{\pi^\alpha}(s) \\
=& E_{\tau\sim\pi^{\alpha'}}[\sum_{t=0}^{\infty}\gamma^t[R^\upsilon(s_t,a_t) - V^{\upsilon}_{\pi^\alpha}(s)] + \sum_{t=0}^{\infty}\gamma^{t+1} V^{\upsilon}_{\pi^\alpha}(s_{t+1})] \\
=& E_{\tau\sim\pi^{\alpha'}}[\sum_{t=0}^{\infty}\gamma^t[R^\upsilon(s_t,a_t) - V^{\upsilon}_{\pi^\alpha}(s) + \gamma V^{\upsilon}_{\pi^\alpha}(s_{t+1})]] \\
=& E_{\tau\sim\pi^{\alpha'}}[\sum_{t=0}^{\infty}\gamma^t[R^\upsilon(s_t,a_t) - V^{\upsilon}_{\pi^\alpha}(s) + \gamma E_{s_{t+1}|s_t,a_t}[V^{\upsilon}_{\pi^\alpha}(s_{t+1})]]] \\
=& E_{\tau\sim\pi^{\alpha'}}[\sum_{t=0}^{\infty}\gamma^t[R^\upsilon(s_t,a_t) + \gamma E_{s_{t+1}|s_t,a_t}[V^{\upsilon}_{\pi^\alpha}(s_{t+1})] - V^{\upsilon}_{\pi^\alpha}(s)]] \\
=& E_{\tau\sim\pi^{\alpha'}}[\sum_{t=0}^{\infty}\gamma^t[Q^{\upsilon}_{\pi^\alpha}(s_t,a^\alpha_t) - V^{\upsilon}_{\pi^\alpha}(s)]] \\
=& E_{\tau\sim\pi^{\alpha'}}[\sum_{t=0}^{\infty}\gamma^t A^{\upsilon}_{\pi^\alpha}(s_t,a^\alpha_t)]\,,
\end{aligned} \tag{10}$$

where $a_t = (a^\alpha_t, F^{\pi^\upsilon}(s_t)))$.

S3 Proof of Theorem 1

Theorem 1. *The difference between $V^{\upsilon}_{\pi^{\alpha'}}(s)$ and $L^{\upsilon}_{\pi^\alpha}(\pi^{\alpha'})$ is bounded by:*

$$\begin{aligned}
V^{\upsilon}_{\pi^{\alpha'}}(s) \le L^{\upsilon}_{\pi^\alpha}(\pi^{\alpha'}) + C_2\mathbb{KL}^{\max}(\pi^\alpha||\pi^{\alpha'}) = M^{\upsilon}_{\pi^\alpha}(\pi^{\alpha'})\,, \\
C_2 = \frac{4\max_{s,a^\alpha}|A^{\upsilon}_{\pi^\alpha}(s,a^\alpha)|\gamma}{(1-\gamma)^2}\,.
\end{aligned} \tag{11}$$

Proof.

$$\begin{aligned}
&V^{\upsilon}_{\pi^{\alpha'}}(s) - L^{\upsilon}_{\pi^{\alpha}}(\pi^{\alpha'}) \\
=&\sum_s P(s_t = s|\pi^{\alpha'}) \sum_a \pi^{\alpha'}(a^{\alpha}|s) \sum_t \gamma^t A^{\upsilon}_{\pi^{\alpha}}(s, a^{\alpha}) - \sum_s P(s_t = s|\pi^{\alpha}) \sum_a \pi^{\alpha'}(a^{\alpha}|s) \sum_t \gamma^t A^{\upsilon}_{\pi^{\alpha}}(s, a^{\alpha}) \\
=&\mathbb{E}_{s_t\sim\pi^{\alpha'}}[\sum_t \gamma^t \mathbb{E}_{a\sim\pi^{\alpha'}(\cdot|s)}[A^{\upsilon}_{\pi^{\alpha}}(s, a^{\alpha})]] - \mathbb{E}_{s_t\sim\pi^{\alpha}}[\sum_t \gamma^t \mathbb{E}_{a\sim\pi^{\alpha'}(\cdot|s)}[A^{\upsilon}_{\pi^{\alpha}}(s, a^{\alpha})]] \\
=&\sum_t \gamma^t [\mathbb{E}_{s_t\sim\pi^{\alpha'}}[\bar{A}^{\upsilon}_{\pi^{\alpha'}}(s_t)] - \mathbb{E}_{s_t\sim\pi^{\alpha}}[\bar{A}^{\upsilon}_{\pi^{\alpha'}}(s_t)]] \,,
\end{aligned} \tag{12}$$

where $\bar{A}^{\upsilon}_{\pi^{\alpha'}}(s) = \mathbb{E}_{a\sim\pi^{\alpha'}(\cdot|s)}[A^{\upsilon}_{\pi^{\alpha}}(s, a^{\alpha})]$.

Let n_t denotes the number of time steps that $a_i^{\alpha'} \neq a_i^{\alpha}$ for time step $i < t$, where $a_i^{\alpha'} \sim \pi^{\alpha'}$ and $a_i^{\alpha} \sim \pi^{\alpha}$. That is, the number of time steps that $\pi^{\alpha'}$ and π^{α} disagrees before time step t.

Then,

$$\mathbb{E}_{s_t\sim\pi^{\alpha'}}[\bar{A}^{\upsilon}_{\pi^{\alpha'}}(s_t)] - \mathbb{E}_{s_t\sim\pi^{\alpha}}[\bar{A}^{\upsilon}_{\pi^{\alpha'}}(s_t)] = P(n_t > 0)(\mathbb{E}_{s_t\sim\pi^{\alpha'}|n_t>0}[\bar{A}^{\upsilon}_{\pi^{\alpha'}}(s_t)] - \mathbb{E}_{s_t\sim\pi^{\alpha}|n_t>0}[\bar{A}^{\upsilon}_{\pi^{\alpha'}}(s_t)])\,. \tag{13}$$

Given that $(\pi^{\alpha'}, \pi^{\alpha})$ is an $\beta-$coupled policy pair [11], we have

$$P(a_i^{\alpha'} = a_i^{\alpha}) \geq 1 - \beta\,. \tag{14}$$

We change the original notation α to β to avoid confusion with the α defined in our paper (*i.e.,* the adversarial agent). Then, we have

$$p(n_t = 0) = \prod_{i=1}^{t} P(a_i^{\alpha'} = a_i^{\alpha}) \geq (1-\beta)^t\,, \tag{15}$$

and $p(n_t > 0) \leq 1 - (1-\beta)^t$. Then, we can derive

$$\begin{aligned}
&\mathbb{E}_{s_t\sim\pi^{\alpha'}}[\bar{A}^{\upsilon}_{\pi^{\alpha'}}(s_t)] - \mathbb{E}_{s_t\sim\pi^{\alpha}}[\bar{A}^{\upsilon}_{\pi^{\alpha'}}(s_t)] \\
=&P(n_t > 0)(\mathbb{E}_{s_t\sim\pi^{\alpha'}|n_t>0}[\bar{A}^{\upsilon}_{\pi^{\alpha'}}(s_t)] - \mathbb{E}_{s_t\sim\pi^{\alpha}|n_t>0}[\bar{A}^{\upsilon}_{\pi^{\alpha'}}(s_t)]) \\
\leq&P(n_t > 0)(|\mathbb{E}_{s_t\sim\pi^{\alpha'}|n_t>0}[\bar{A}^{\upsilon}_{\pi^{\alpha'}}(s_t)]| + |\mathbb{E}_{s_t\sim\pi^{\alpha}|n_t>0}[\bar{A}^{\upsilon}_{\pi^{\alpha'}}(s_t)]|) \\
\overset{(a)}{\leq}&P(n_t > 0)4\beta\max_{s,a^{\alpha}}|A^{\upsilon}_{\pi^{\alpha}}(s, a^{\alpha})| \\
\leq&(1 - (1-\beta)^t)4\beta\max_{s,a^{\alpha}}|A^{\upsilon}_{\pi^{\alpha}}(s, a^{\alpha})|\,.
\end{aligned} \tag{16}$$

Where (a) can be obtained based on the following relationship. First, given that $\mathbb{E}_{a^{\alpha}\sim\pi^{\alpha}}[A^{\upsilon}_{\pi^{\alpha}}(s_t, a^{\alpha})] = 0$, we have

$$\begin{aligned}
\bar{A}^{\upsilon}_{\pi^{\alpha'}}(s_t) =&\mathbb{E}_{a^{\alpha'}\sim\pi^{\alpha'}(\cdot|s_t)}[A^{\upsilon}_{\pi^{\alpha}}(s_t, a^{\alpha'})] \\
=&P(a^{\alpha} \neq a^{\alpha'})\mathbb{E}_{(a^{\alpha},a^{\alpha'})\sim(\pi^{\alpha},\pi^{\alpha'})|a^{\alpha}\neq a^{\alpha'}}[A^{\upsilon}_{\pi^{\alpha}}(s_t, a^{\alpha'}) - A^{\upsilon}_{\pi^{\alpha}}(s_t, a^{\alpha})]\,.
\end{aligned} \tag{17}$$

Based on Eqn. (17), we can further derive that

$$|\bar{A}^{\upsilon}_{\pi^{\alpha'}}(s_t)| \leq P(a^{\alpha} \neq a^{\alpha'})\mathbb{E}[|A^{\upsilon}_{\pi^{\alpha}}(s_t, a^{\alpha'})| + |A^{\upsilon}_{\pi^{\alpha}}(s_t, a^{\alpha})|] \leq \beta \cdot 2\max_{s,a^{\alpha}}|A^{\upsilon}_{\pi^{\alpha}}(s, a^{\alpha})|\,. \tag{18}$$

Given that $\bar{A}^{\upsilon}_{\pi^{\alpha'}}$ at any state fulfills the inequality in Eqn. (18), the expectation of $\bar{A}^{\upsilon}_{\pi^{\alpha'}}$ over all the states also obeys this inequality. As such, we have

$$|\mathbb{E}_{s_t\sim\pi^{\alpha}|n_t>0}[\bar{A}^{\upsilon}_{\pi^{\alpha'}}(s_t)]| \leq \beta \cdot 2\max_{s,a^{\alpha}}|A^{\upsilon}_{\pi^{\alpha}}(s, a^{\alpha})|\,. \tag{19}$$

According to Eqn. (19), we can have the (a) in Eqn. (16).

Plugging Eqn. (16) into Eqn. (12), we have

$$\begin{aligned}
V^{\upsilon}_{\pi^{\alpha'}}(s) - L^{\upsilon}_{\pi^{\alpha}}(\pi^{\alpha'}) =&\sum_t \gamma^t [\mathbb{E}_{s_t\sim\pi^{\alpha'}}[\bar{A}^{\upsilon}_{\pi^{\alpha'}}(s_t)] - \mathbb{E}_{s_t\sim\pi^{\alpha}}[\bar{A}^{\upsilon}_{\pi^{\alpha'}}(s_t)]] \\
\leq&\sum_t \gamma^t (1 - (1-\beta)^t)4\beta\max_{s,a^{\alpha}}|A^{\upsilon}_{\pi^{\alpha}}(s, a^{\alpha})| \\
=&4\varepsilon\beta\sum_t \gamma^t (1 - (1-\beta)^t) \\
=&\frac{4\varepsilon\gamma\beta^2}{(1-\gamma)(1-\gamma(1-\beta))} \\
\leq&\frac{4\varepsilon\gamma\beta^2}{(1-\gamma)^2}\,,
\end{aligned} \tag{20}$$

where $\varepsilon = \max_{s,a^\alpha} |A^v_{\pi^\alpha}(s, a^\alpha)|$. According to a Proposition in [5], $\beta = \max_s \mathbb{TV}(\pi^\alpha(\cdot|s)||\pi^{\alpha'}(\cdot|s))$. Then, we can derive

$$V^v_{\pi^{\alpha'}}(s) \leq L^v_{\pi^\alpha}(\pi^{\alpha'}) + \frac{4\varepsilon\gamma\beta^2}{(1-\gamma)^2} \,. \tag{21}$$

According to [10], $\mathbb{TV}(p||q)^2 \leq \mathbb{KL}(p||q)$. Plugging this relationship into Eqn. (21), we have

$$V^v_{\pi^{\alpha'}}(s) \leq L^v_{\pi^\alpha}(\pi^{\alpha'}) + C_2 \mathbb{KL}^{\max}(\pi^\alpha(\cdot|s)||\pi^{\alpha'}(\cdot|s)) \,, \tag{22}$$

where $C_2 = \frac{4\gamma \max_{s,a^\alpha} |A^v_{\pi^\alpha}(s,a^\alpha)|}{(1-\gamma)^2}$. □

S4 Adversarial Learning Algorithm

In this section, we first describe how to transform the approximated objective function $M_{\pi^\alpha}(\pi^{\alpha'})$ into our final adversarial learning objective function, followed by our adversarial learning algorithm.

Here, we first rewrite the Eqn. (11) in Section 3.3,

$$M_{\pi^\alpha}(\pi^{\alpha'}) = \sum_s \rho_{\pi^\alpha}(s) \sum_a \pi^{\alpha'}(a^\alpha|s)(A^\alpha_{\pi^\alpha}(s,a^\alpha) - A^v_{\pi^\alpha}(s,a^\alpha)) - C\mathbb{KL}^{\max}(\pi^\alpha||\pi^{\alpha'}) + C_3 \,, \tag{23}$$

where $C = C_1 - C_2$ and $C_3 = (V^\alpha_{\pi^\alpha}(s) - V^v_{\pi^\alpha}(s))$ are constants. Then, by following the method introduced in TRPO, we can further transform the maximization of Eqn. (23) into the following form

$$\begin{aligned} &\text{maximize}_{\pi^{\alpha'}} \sum_s \rho_{\pi^\alpha}(s) \sum_a \pi^{\alpha'}(a^\alpha|s)(A^\alpha_{\pi^\alpha}(s,a^\alpha) - A^v_{\pi^\alpha}(s,a^\alpha)) \,, \\ &\text{s.t. } \mathbb{E}_{s\sim\pi^\alpha}[\mathbb{KL}(\pi^\alpha(\cdot|s)||\pi^{\alpha'}(\cdot|s))] \leq \delta \,. \end{aligned} \tag{24}$$

As we can see from the equation above, this transformation replaces $\mathbb{KL}^{\max}$ $(\pi^\alpha||\pi^{\alpha'})$ in Eqn. (23) with $\mathbb{E}_{s\sim\pi^\alpha}[\mathbb{KL}(\pi^\alpha||\pi^{\alpha'})]$ for the following reasons. $\mathbb{E}_{s\sim\pi^\alpha}$ $[\mathbb{KL}(\pi^\alpha||\pi^{\alpha'})]$ is the average KL divergence between π^α and $\pi^{\alpha'}$, which can be easily computed by using the Monte Carlo method [14]. Using this expectation as the substitution for maximum KL divergence, it is no longer required us to perform intensive computation at each state. In addition to the computation benefit, the replacement of maximum KL divergence indicates the ease of solving optimization. When performing optimization with maximum KL divergence, we have to introduce a constraint for each state. Given a two-player game with many states, this means imposing a large number of constraints on our optimization problem and potentially introduces the difficulty in getting an optimal solution. Note that, as is experimented in [9, 8], applying such an approximation imposes only a minor variation to the resolved policy.

As we can also observe from Eqn. (23), in Eqn. (24), we also transform the term maximize $-C\mathbb{KL}^{\max}(\pi^\alpha||\pi^{\alpha'})$ into a trust region constraint $\mathbb{E}[\mathbb{KL}(\pi^\alpha||\pi^{\alpha'})] \leq \delta$. [1] As is discussed in [11], this transformation could enable a larger step size for the optimization process and thus accelerate the optimization process.

Even with all the transformation above, optimizing Eqn. (24) is still challenging. As we can see, this optimization objective involves the computation of $\sum_a \pi^{\alpha'}(a^\alpha|s)$ which contains the actions tied to the new policy $\pi^{\alpha'}$. Before getting the optimization result, these actions are unknown. As a result, computing $\sum_a \pi^{\alpha'}(a^\alpha|s)(A^\alpha_{\pi^\alpha}(s,a^\alpha) - A^v_{\pi^\alpha}(s,a^\alpha))$ is intractable.

To solve this problem, we again follow the idea of TRPO, apply an important sampling estimator

$$\begin{aligned} &\sum_a \pi^{\alpha'}(a^\alpha|s)(A^\alpha_{\pi^\alpha}(s,a^\alpha) - A^v_{\pi^\alpha}(s,a^\alpha)) \\ =&\sum_a \frac{\pi^{\alpha'}(a^\alpha|s)}{\pi^\alpha(a^\alpha|s)} \pi^\alpha(a^\alpha|s)(A^\alpha_{\pi^\alpha}(s,a^\alpha) - A^v_{\pi^\alpha}(s,a^\alpha)) \\ =&E_{a\sim\pi^\alpha}[\frac{\pi^{\alpha'}(a^\alpha|s)}{\pi^\alpha(a^\alpha|s)}(A^\alpha_{\pi^\alpha}(s,a^\alpha) - A^v_{\pi^\alpha}(s,a^\alpha))] \,, \end{aligned} \tag{25}$$

[1] $C = \frac{4\gamma(\max_{s,a^\alpha}|A^\alpha_{\pi^\alpha}(s,a^\alpha)| - \max_{s,a^\alpha}|A^v_{\pi^\alpha}(s,a^\alpha)|)}{(1-\gamma)^2}$. Given that, in each iteration, our optimization maximizes the value function difference between the adversary and victim, we can obtain $C > 0$ for the current policy π^α.

Algorithm 1: Adversarial learning algorithm.

```
1  Input: The adversarial policy $\pi_\theta^\alpha$ parameterized by $\theta$, the state-value function $V_{\pi^\alpha}^{\alpha}$ and $V_{\pi^\alpha}^{\upsilon}$,
2            with parameters $v_\alpha$ and $v_\upsilon$, respectively.
3  Initialization: Initialize $\theta^{(0)}$, $v_\alpha^{(0)}$ and $v_\upsilon^{(0)}$.
4  for $k = 0, 1, 2, ..., K$ do
5      Use the current adversarial policy $\pi_{\theta^{(k)}}^\alpha$ to play with the victim agent with a fixed policy
         $\pi^v$, and collect a set of trajectories $\mathcal{D}(k) = \{\tau_i\}$, where $i = 1, 2, ...., |\mathcal{D}^{(k)}|$.
6      For each trajectory $\tau_i$, compute the advantage at each time step $t$ ($t = 0, 1, 2, ..., |\tau_i|$):
7      $A_{i_t}^{\alpha(k)} = r_{it}^{\alpha(k)} + \gamma V^{\alpha(k)}(o_{i_{t+1}}^{\alpha(k)}) - V^{\alpha(k)}(o_{i_t}^{\alpha(k)})$;
8      $A_{i_t}^{\upsilon(k)} = r_{i_t}^{\upsilon(k)} + \gamma V^{\upsilon(k)}(o_{i_{t+1}}^{\alpha(k)}) - V^{\upsilon(k)}(o_{i_t}^{\alpha(k)})$,
9      where we omit the subscript $\pi_{\theta^{(k)}}^\alpha$ for simplicity.
10     Introduce $A_{i_{0:|\tau_i|}}^{\alpha(k)}$ and $A_{i_{0:|\tau_i|}}^{\upsilon(k)}$ ($i = 1 : |\mathcal{D}^{(k)}|$) into Eqn. (27) and obtain a new policy by
         maximizing the objective function in Eqn. (27).
11     Update $v_\alpha^{(k)}$ and $v_\upsilon^{(k)}$ by solving argmin$\frac{1}{T}\sum_{t=0}^{T}(V(o_t) - (r_t + \gamma V_{old}(o_{t+1})))^2$.
12 end
13 Output: The adversarial policy network $\pi_\theta^\alpha$.
```

and thus transform Eqn. (24) into the form of

$$\begin{aligned}&\text{argmax}_\theta \; \mathbb{E}_{\pi_{old}^\alpha}\Big[\frac{\pi_\theta^\alpha(a_t^\alpha|s_t)}{\pi_{old}^\alpha(a_t^\alpha|s_t)}(A_{\pi_{old}}^{\alpha}(a_t^\alpha, s_t) - A_{\pi_{old}}^{\upsilon}(a_t^\alpha, s_t))\Big]\,,\\ &s.t.\; \mathbb{E}_{s_t \sim \pi_{old}^\alpha}[\mathbb{KL}(\pi_{old}^\alpha(\cdot|s_t)||\pi_\theta^\alpha(\cdot|s_t))] \leq \delta\,.\end{aligned} \tag{26}$$

As we can observe from the equation above, the expectation does not rely upon the actions pertaining to the new policy (π_θ^α) but those tied to the old one (π_{old}^α). To learn an adversarial policy, we can optimize this objective function by using the algorithm introduced in TRPO [11]. However, in order to further improve the efficiency and effectiveness of the learning process, we follow the PPO algorithm [12], apply the clipped ratio operation, and obtain the following optimization function

$$\begin{aligned}&\text{argmax}_\theta \; \mathbb{E}_{(a_t^\alpha, s_t)\sim \pi_{old}^\alpha}[\min(\text{clip}(\rho_t, 1-\epsilon, 1+\epsilon)A_t^{\alpha}, \rho_t A_t^{\alpha}) - \min(\text{clip}(\rho_t, 1-\epsilon, 1+\epsilon)A_t^{\upsilon}, \rho_t A_t^{\upsilon})]\,,\\ &\rho_t = \frac{\pi_\theta^\alpha(a_t^\alpha|s_t)}{\pi_{old}^\alpha(a_t^\alpha|s_t)}, A_t^{\alpha} = A_{\pi_{old}^\alpha}^{\alpha}(a_t^\alpha, s_t), A_t^{\upsilon} = A_{\pi_{old}^\alpha}^{\upsilon}(a_t^\alpha, s_t)\,.\end{aligned} \tag{27}$$

In this work, we use this equation as our ultimate objective function and follow the procedure below to resolve this objective. Algorithm 1 shows our proposed adversarial learning algorithm. Specifically, we first approximate the corresponding state-value functions by using two deep neural networks, at the time step t, each of which takes as input the adversarial observation o_t^α and outputs the approximated value for the state-value function V_t^α and V_t^υ. Second, we compute the parameters of these two networks by solving the optimization function in line 11. With the parameters resolved, we further update the adversarial policy network by solving the optimization function in Eqn. (27). In each training iteration, we gather a set of training trajectories by using the current adversarial agent to play with the fixed victim agent. By using the collected trajectories, we update the adversarial policy network and the networks pertaining to the two state-value functions. Note that, compared with the PPO algorithm used in the state-of-art attack [4], our proposed learning algorithm trains one additional value function and solves a more complicated optimization function. This leads to extra computational cost. To estimate this extra cost, we use the same machine (a server with 32 CPUs) to run our method and the state-of-art attack and record their runtimes. The average runtime of our method is about 1.4X over that of the existing attack (*e.g.,* 20 hours vs. 16 hours on the You-Shall-Not-Pass game, 32 hours vs. 23 hours on the Kick-And-Defend game). Considering the significant improvement in exploitability, we believe that this amount of extra cost is acceptable.

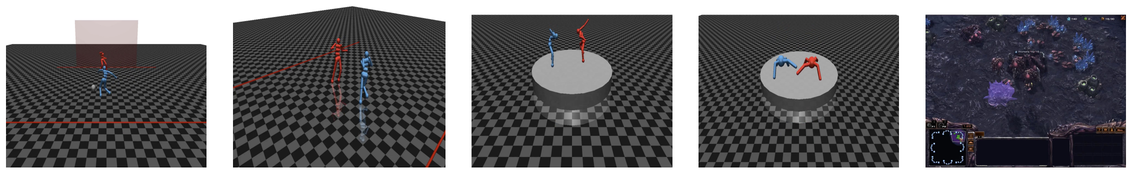

(a) Kick And Defend. (b) You Shall Not Pass. (c) Sumo Humans. (d) Sumo Ants. (e) StarCraft II.

Figure S1: The snapshots of the two-player games selected for our experiment. The first four games are from MuJoCo game zoo and the last one is StarCraft II.

S5 Implementation and Experiment Setups

S5.1 Implementation and Hyper-parameter selection

We implemented our learning algorithm based on the `TensorFlow` [1] and `Stable Baselines` [7] packages. We implemented the baseline attack [4] based on the code released by the authors: `https://github.com/HumanCompatibleAI/adversarial-policies`. We also implemented the game environment wrappers based on the OpenAI Gym (*i.e.,* `https://gym.openai.com/`), Multi-agent Competition (*i.e.,* `https://github.com/openai/multiagent-competition`), as well as PySC2 Extension [13] packages.

In the following, we specify the choices of hyper-parameters for our attack and the state-of-art attack [4]. Both attacks have two sets of hyper-parameters: policy network/value function architectures and learning algorithm hyper-parameters (*e.g.,* clipping parameter ϵ, discount factor γ, learning rate). To ensure a fair comparison, we adopted the same set of hyper-parameters in these two attacks. To be specific, for the adversarial policy network/state-value function architecture, we followed the choice in [4] and set them as Multilayer Perceptron with different layers for different games. The details of the network architectures can be found in `https://github.com/HumanCompatibleAI/adversarial-policies`. Regarding the learning algorithm hyper-parameters, we also used the default choice of the state-of-art attack [4]. The exact values are also shown in `https://github.com/HumanCompatibleAI/adversarial-policies`. Regarding the adversarial retraining experiments, we directly retrained the original victim agents with their original policy network/state-value function architectures and the same set of training hyper-parameters used in the attack experiments.

For the StarCraft II game, we also used the same set of hyper-parameters for these two attacks. Specifically, we directly adopted the default choices released by the `PySC2 Extension` platform, upon which we train the RL agents. The network architectures are also Multilayer Perceptrons. The detailed network architectures and the value of the training hyper-parameters can be found in `https://github.com/Tencent/TStarBot1`. It should be noted that compared to state-of-the-art attack, the only additional hyper-parameters introduced by our attack is the weight between the first and second term in our learning objective function. We varied the weight of the second term between [1, 4] and found that this variation imposes only a minor change upon the exploitability and transferability of our attack. As such, we gave these two terms the equal weight in our experiments.

S5.2 Experiment Setups

Game selection & obtaining victim agents. We select five games for our experiments, including four robotic games from MuJoCo game zoo [15] and one real-time strategy game – StarCraft II [16]. Researchers commonly adopt these games in academia and industry to evaluate reinforcement learning algorithms in a two-player game context (*e.g.,* [2, 4, 6, 13]). For each of the games, researchers have released many benchmark game bots [2, 16]. Concerning the bots designed for MuJoCo games, they are all trained through DRL. However, the policy networks used in the bots are different (*e.g.,* "Sumo-Humans" and You-Shall-Not-Pass" use LSTM and an MLP as their policy networks, respectively). In this work, we use the following criteria to select our victim agent. First, we train an adversarial agent by using an existing technique [4]. Then, we use it to play with each of the agents and record the winning rate. For the agent demonstrating the highest winning rate against the adversary, we choose it as the victim agent of that game. For the agent with the second-highest winning rate, we select it as a regular agent for evaluating the transferability of our adversarial agent and the generalizability of our retrained victim agent. It should be noted that we compute the winning rate by having the corresponding agent play with its opponent for 100 rounds and reporting the number of its wins. It should also be noted that for the asymmetric games "You-Shall-Not-Pass" and "Kick-And-Defend", we select the runner and kicker as the victim agent, respectively.

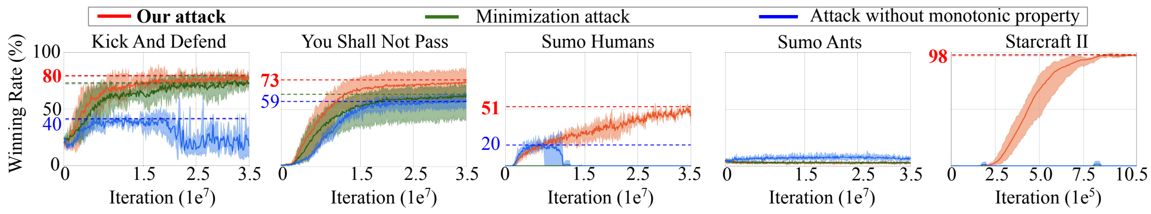
Regarding the real-time strategy game StarCraft II, the game vendor, and their collaborators release seven bots indicating different master levels. (*i.e.,* level-1 to level-7 represents amateur to elite). These bots take actions under the guidance of different sets of pre-defined rules but not a policy network trained with an RL algorithm. As a result, we follow the method proposed by DeepMind [3], use the PPO algorithm to prepare two game agents, and ensure both of our game agents could demonstrate the decisive winning rates (*i.e.,* $> 94\%$) against all the rule-based agents. In this work, we employ one agent as the victim agent and the other as the regular agent for the StarCraft game. In the following, we provide a more detailed description of each of the games mentioned above. Upon the acceptance of this work, we will release our source code and all the agents/environments used for our evaluation.

MuJoCo–Kick And Defend. This is a soccer penalty shootout, in which the kicker (*i.e.,* the blue humanoid robot in Figure 1(a)) intends to shoot the ball into the net (*i.e.,* the grey region on the red line in Figure 1(a)), whereas the defender (*i.e.,* the red humanoid robot in Figure 1(a)) prevents the kicker from scoring the goal. A successful scoring gives the kicker +1000 reward and the defender opponent -1000 reward. On the contrary, A successful defending gives the defender +1000 reward and the kicker -1000 reward. The defender is awarded an additional +500 reward if it saves a penalty and establishes certain desired behaviors. However, if the defender moves out of a defined goalkeeping region during a game, it gets a punishment of -1000 reward, and the game will end as a draw. Note that, in this game, a game episode exceeding the maximum time is treated as a successful defending.

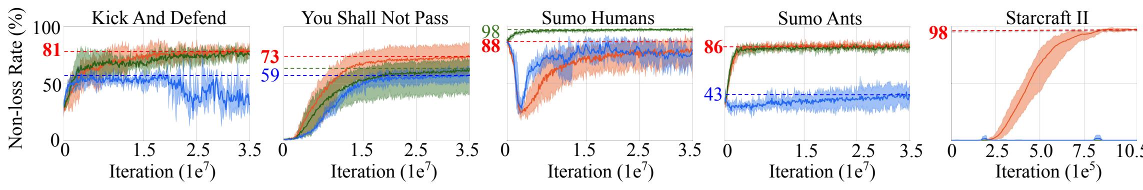
MuJoCo–You Shall Not Pass. As is illustrated in Figure 1(b), the two agents in this game start by facing each other. Then, the blue humanoid robot (*i.e.,* runner) starts to run towards the finish line (*i.e.,* indicated by the red line in Figure 1(b)). Meanwhile, the red humanoid robot (*i.e.,* blocker) attempts to block the blue robot from reaching the finish line. If the red robot successfully stops its opponent and it keeps standing till the end of a game, it could receive +1000 reward. If it blocks its opponent but falls into the ground, it gets 0 reward. In both cases, the blue robot gets -1000 reward. On the contrary, if the blue robot reaches the finish line, it receives +1000 reward, and the red robot gets -1000 reward.

MuJoCo–Sumo Humans and Sumo Ants. In both games, the robots are randomly initialized at different positions on the grey round arena in Figure 1(c) and 1(d). Then, they start to move and push each other. One of the agents wins if it knocks its opponent into the ground or pushes it out of the arena. The winner receivers +1000 reward, and the loser gets -1000 reward. If one agent falls from the arena without contacting its opponent or the game exceeds the maximum time, the game ends with a tie. Different from the games mentioned above, where the agent receives 0 reward in a tie game, in Sumo games, both agents get a penalty of -1000 reward for a draw. As is shown in Figure 1(c) and 1(d), the only difference between Sumo Humans and Sumo Ants is the shape of the robots. Note that, different from the two games introduced above, the agents are symmetric in the Sumo games.

StarCraft II. As is depicted in Figure 1(e), the base of each player is randomly placed at a corner on the map. Then, the players start to take action according to their strategies. The goal for each player is to defeat its opponent within a limited time. A game exceeding the time limit ends as a tie. In this paper, we train the reinforcement learning agents (players) on the `PySC2 Extension` platform released by [13]. To be specific, it designs 165 macro actions for an agent, each of which is a combination of the original operations in StarCraftII games. These macro actions can be categorized into five types – collecting resources, constructing buildings, producing workers and solders, upgrading technology, and combating. More details about the macro actions can be found in [13]. At the end of a game, each agent receives a reward based on the game result: 1 (win), 0 (tie), and -1 (lose). During the game, they also receive some additional rewards based on the number of enemies they have killed and the amount of resources they collected. Similar to [13], we consider a two-player competitive full-game in our experiments, in which both players belong to Zerg. Training an RL agent on a real gaming map



(a) The adversarial winning rates. Note that the green line in StarCraft II is overlapped with the blue line.



(b) The adversarial non-loss rates. The winning+tie rate of minimization attack is always zero in StarCraft II.

Figure S2: The performance comparison of adversarial agents trained with our attack and other two comparison methods: the attack that only minimizes the victim value function indicated by "Minimization attack" (*i.e.,* green lines and shadows in the figure) and the attack without monotonic property (*i.e.,* blue lines and shadows in the figure).

Table S1: Victim agents' performances against our adversarial agents before retraining.

	Kick And Defend	You Shall Not Pass	Sumo Humans	Sumo Ants	StarCraft II
Winning (%)	25.0	30.0	30.0	15.0	2.0
Non-loss (%)	26.0	30.0	63.0	97.0	2.0

requires a large amount of time and computational resources. It takes [13] about two days and more than 3,000 CPUs to train an agent in a real gaming map. Due to limited computation resources, we use a smaller map designed specifically for training RL agents [16] rather than the real gaming maps. As is demonstrated in [6], the results of the smaller maps can be generalized to the real one by training the agents for a longer time.

(Re)training & performance quantification. From Algorithm 1, we could quickly note that, in each of the training iterations, our adversarial agent interacts with the victim, collects trajectories, and updates its policy network accordingly. To reduce the impact of state randomness (*e.g.,* the agent's initial position on the map and the probabilistic state transitions) upon our adversarial agent's performance, we follow the previous research [2, 4] and repeat each experiment six times by randomly selecting different initial states. With this setup, we report the average performance of our adversarial agents as well as their performance variance. Also, it should be noted that the algorithm updates our policy at each iteration. Therefore, we report the average performance of an adversarial agent every time its policy is updated. In our training process, we keep updating our adversarial agent iteratively until the agent performance converges. For all MuJoCo games, in the training process, the adversarial agent performance converges after 35 million iterations. For StarCraft II, it plateaus after 1.05 million iterations. As we will discuss below, we also design an experiment to evaluate the effectiveness of adversarial retraining for victim agents. For MuJoCo and StarCraft games, in the process of victim agent retraining, agent performance stays stable after 10 million and 2.2 million iterations, respectively. Similar to the setup for learning an adversarial agent, when retaining a victim agent, we follow the same setup process, which repeats our experiment for six times and reports average performance accordingly.

S6 Additional Experiments.

Significance of performance difference. Section 4.1 visualizes the winning rate comparison between our adversarial agents and that obtained by the baseline approach. Here, we also conduct

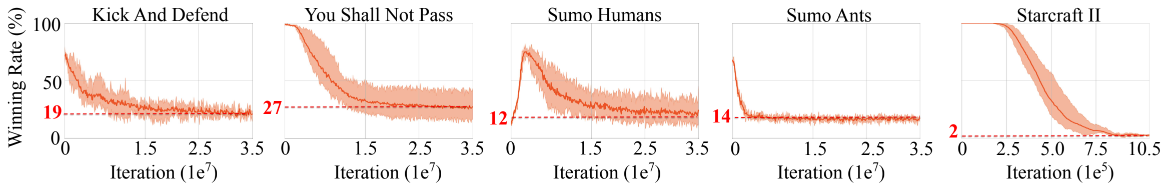


Figure S3: The changes in the winning rates of victim agents when training our adversarial agents against them.

Table S2: Mean, std and the p-value of the winning rate diff.

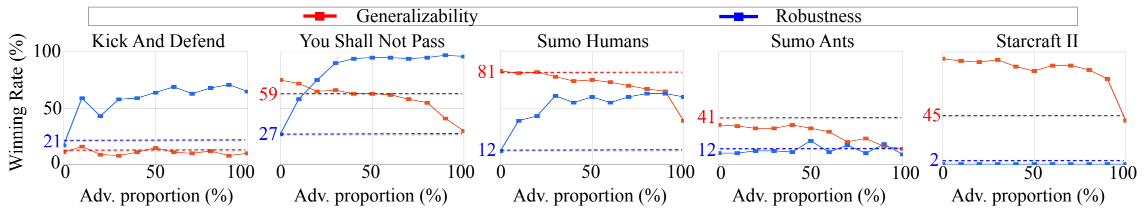
	Kick And Defend	You Shall Not Pass	Sumo Humans	Sumo Ants	StarCraft II
Mean/Std (%)	8.2/2.8	24.1/9.4	14.6/9.0	37.5/6.2 (Non-loss rate)	56.9/13.2
P-value	0.002	0.003	0.007	<0.001 (Non-loss rate)	0.002

a statistical measure on the winning rate difference between our attack (s_o) and the baseline (s_b). Specifically, we first compute their diff ($D = s_o - s_b$) in each run. Then, we compute the mean, std, and the p-value of the paired t-test. For the t-test, our null hypothesis is $H_0 : \mathbb{E}[D] \leq 0$. If p-value is larger than an empirical threshold (*e.g.,* 0.01), we accept H_0, indicating our method cannot outperform the baseline. The results in Table S2 indicate a rejection of H_0, confirming our method's superiority over the baseline approach.

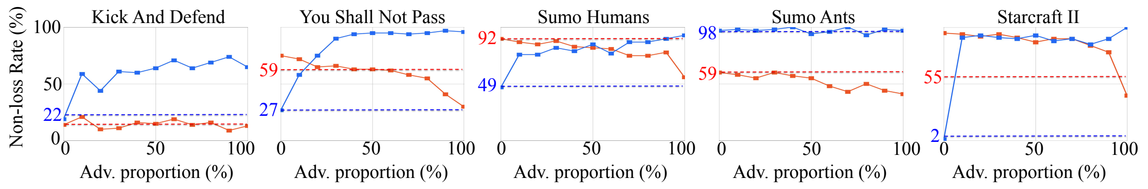
Our attack vs. other comparison methods. In this experiment, we compare our attack with two other methods of training an adversarial agent. We first consider an adversarial learning algorithm that optimizes a similar objective with our attack but without monotonic property. As is discussed in Section 3.2, a trivial way to solve the our proposed objective function is to approximate the second term in our learning objective with a DNN $G^{v}_{\pi^{\alpha}}(\cdot)$ and combine it with the TRPO objective function (*i.e.,* $M^{\alpha}_{\pi^{\alpha}}(\cdot)$). By maximizing this approximation of the objective function i.e., $M^{\alpha}_{\pi^{\alpha}}(\cdot) - G^{v}_{\pi^{\alpha}}(\cdot)$) with stochastic gradient ascent, one could solve an adversarial policy that shares a similar learning goal with our attack. However, this method cannot guarantee a monotonical increase in the difference between the expected total rewards of the adversarial agent and the victim agent. In this experiment, we apply this trivial solution to the selected games and compare the performance of the trained agent with the adversarial agent prepared by our attack.

We also compare our attack with another method that utilizes only the second term in our final learning objective function as the objective function (*i.e.,* $\text{argmax}_{\theta} -\mathbb{E}[\min(\text{clip}(\rho_t, 1-\epsilon, 1+\epsilon)A^{v}_{t}, \rho_t A^{v}_{t})]$). Recall that our final learning objective function contains two objectives. The first is to maximize the expected total reward of the adversarial agent, whereas the second is to minimize that of the victim. By setting up this experiment, we study the effect of the second objective upon our attack against the victim because intuition suggests the minimization of victim reward alone could also downgrade the performance of the victim and thus lead the potential victory of the adversary.

While this objective is distinct from that of (Gleave et.al 2020), it is my impression that the approach of (Gleave et.al 2020) is subtly mischaracterized in that it was meant to train policies which are adversarial to a particular policy (in that they minimize the victim's reward), and not self interested (in the sense that they maximize their own reward). In the zero-sum environments these correspond very closely, but in the Starcraft II environment it seems that the suitable baseline would not be to train on the adversary's environment reward, but on the victim's environment reward. This seems to be a reasonable model of "adversarial behavior" even in general-sum games, but many applications it would be unrealistic (for instance adversarial cars following this model would crash into the victim with no regards to their own safety). Changing this aspect of the paper not only would be a more accurate reflection of prior work, but would likely clarify the difference between this approach and prior work. To further clarify this difference I would suggest to give the new objective a name that is not "adversarial" as it does not follow the typical usage of that framing. You use "hostile" at one point, which could be a suitable replacement as it does not imply that the agent is directly in opposition to the victim.



(a) The winning rates of the retrained victim agents.



(b) The non-loss rates of the retrained victim agents.

Figure S4: The performance of the victim agents retrained with different proportions of normal/adversarial episodes in the retraining episodes. The solid blue lines represent the average winning (plus tie) rates when playing with our adversarial agent (indicated by "robustness"). The solid red lines represent the average winning (plus tie) rates when playing with the second-strongest regular agent (indicated by "generalizability"). The blue and red dash lines represent the robustness and generalizability of the victim agent before the adversarial retraining, respectively. The x-axis label "Adv. proportion" denotes the proportions of adversarial episodes.

We conducted an experiment making adversary focus on minimizing victim reward without caring its own. Supplementary S6 shows the results (minimization attack in Fig. S2). We found, sometimes, this minimization method works but can't outperform ours. For StarCraft II, the minimization method completely fails. This aligns with the reviewer's thoughts. We will emphasize this point in the next version.

Figure S2 shows the performance of the adversarial agent obtained by our attack and the two comparison methods introduced above. In Figure S2, we can first observe that, without the monotonic guarantee, the adversarial winning rate dramatically drops on all of the five games. In all the games except "You-Shall-Not-Pass", the attack without monotonicity performs even worse than the state-of-art attack [4]. This result shows that, without the monotonic guarantee, the newly added minimization term often imposes even a negative impact upon the adversarial learning process and thus result in an adversarial policy with a weaker exploitability than that obtained by the attack without the minimization term. It should be noted that, in the complicated StarCraft II game, the learning process completely fails, resulting in zero winning plus tie rate. This indicates that enabling a monotonic guarantee is especially crucial for sophisticated games.

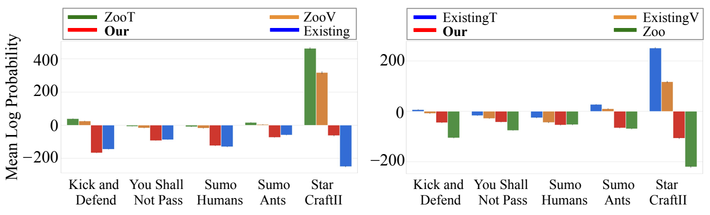
As we can also observe from Figure S2, by taking only the minimization into account alone, the adversarial agents trained do not demonstrate a similar winning rate as our proposed method. However, as is shown in Figure 2(b), they generally exhibit a powerful ability to prevent the victim from winning the game. As we can observe from the games "Sumo Ants and Humans", the ability to downgrade the winning rate of the victim is almost as same as or even better than our proposed method. It indicates that the expected reward minimization could play a critical role in restricting the victim agent's win.

However, from Figure S2, we can also observe the minimization alone does not make any significant difference for StarCraft game. It can barely bring any game wins or ties. We believe the reason is the design of the game. At the beginning of the StarCraft game, the game engine starts both agents at two different corners on a large map. As such, an adversarial agent in the StarCraft game cannot quickly interact with the victim, influence its actions, and thus curtail the victim's reward collection. Instead, to prevent the victim from collecting resources and building up a strong army, the adversarial agent has to first spend time exploring the map and navigating to the base station of the victim. In this period, the victim usually has already constructed an army which is sufficiently strong to beat intruders. Using our approach, which combines both minimizing victim's reward and maximizing the adversarial reward, we can train an agent to have the ability first to gather resources to build up an army and then intervene in the army construction of the victim. With this learned policy, we can eventually obtain decisive wins.

Another point regarding the minimization attack is that, compared to the baseline approach, the policy trained by this attack is more likely to be *adversarial*. This is because rather than being self-interested by maximizing its own reward, the adversarial agent focuses on disturbing the victim agent. However, this attack can be unrealistic in many applications where self-interest is essential for obtaining a feasible policy (*e.g.,* StarCraft and adversarial self-driving cars – following this model would crash into the victim without considering their own safety).

Adversarial retraining with different episode splits. In Section 4.2, we follow the setup in [4] and set the adversarial and normal episodes evenly when conducting the adversarial retraining against our attack. In this experiment, we study the influence of episode split upon the retraining performance. Specifically, we vary the proportion of the adversarial episodes from 0.0% $\sim$ 100.0% by increasing 10% each time and retrain the victim agent by using each episode split. Figure S4 shows the robustness and the generalizability of the victim agent retrained under these settings. As we can first observe from the figure, overall, a higher proportion of adversarial episode enables better robustness, but worse generalizability. As is also shown in the figure, adversarial retraining has different effects on different games. For example, in You-Shall-Not-Pass and Kick-And-Defend, the adversarial retraining always improves the robustness of the retrained victim agent, even with only 10% of adversarial episodes. It should be noted that, if one intends to select a setting, where both the robustness and generalizability are improved, different games will give different results. This indicates there is no universal optimal episode split, and the user has to find the best solution for each game individually. It should also be noted that, in Sumo-Ants, no matter how to vary the episode split, the robustness of the retrained victim keeps almost unchanged, which means adversarial retraining cannot robustify the victim agent against our attack. This result supports that, in this game, our adversarial policy exploits the game unfairness rather than the weakness of the opponent policy.

Adversarial Policy Behavior Analysis. In addition to drawing the demo videos, we also follow [4] and conducts two additional experiments on the selected games to analyze the behavior of our adversarial policies. Specifically, we first mask a victim agent (*i.e.,* zero out the part of the victim observation that corresponds to the adversarial position) and play it with our adversarial agent and that obtained by the existing attack in the corresponding game. Note that the adversarial agents are trained against the original unmasked victims. Table S3 records the victim's winning and non-loss rate before and after masking. As we can first observe from the Table, similar to the findings in [4], the victim winning rates against both attacks increase dramatically in the You-Shall-Not-Pass and Kick-And-Defend game. This result also matches our observations from demo videos that an adversarial agent fails a victim by triggering adversarial observations rather than winning in a regular way. However, as is shown in Section 4, despite establishing similar adversarial behaviors, our adversarial agent has a stronger exploitability than that obtained by the



(a) GMM trained on the victim activations when playing against a regular agent.

(b) GMM trained on the victim activations when playing against the existing attack.

Figure S5: The average GMM log likelihood of the victim activations when playing against different opponents. Figure (a) shows the results of using the victim activations collected by playing against a regular agent to train the GMM. "ZooT" and "ZooV" represents the training/validation activations. "Our" and "Existing" represents the activations collected when playing against our attack and the existing attack. Figure (b) shows the results of training the GMM with the victim activations collected by playing against the existing attack ("ExistingT" and "ExistingV" are the training and testing set; "Our" and "Zoo" are our attack and regular agent).

Table S3: The victim winning/non-loss rates against our adversarial agents before and after masking.

			Kick And Defend	You Shall Not Pass	Sumo Humans	SumoAnts	StarCraft II
Our attack	Before masking	Winning (%)	14.0	26.0	22.0	15.0	1.0
		Non-loss (%)	14.0	26.0	53.0	86.0	2.0
	After masking	Winning (%)	94.0	98.0	25.0	13.0	4.0
		Non-loss (%)	94.0	98.0	67.0	87.0	24.0
Existing attack	Before masking	Winning (%)	45.0	48.0	34.0	57.0	18.0
		Non-loss (%)	45.0	48.0	65.0	91.0	64.0
	After masking	Winning (%)	97.0	97.0	12.0	37.0	65.0
		Non-loss (%)	97.0	97.0	68.0	95.0	98.0

Table S4: The winning rates of our adversarial agents against the mediocre and well-trained victims.

		Kick And Defend	You Shall Not Pass	Sumo Humans	Sumo Ants	StarCraft II
Adv. winning rate	Against mediocre victims (%)	88.0	93.0	55.0	86.0 (Non-loss rate)	98.0
	Against well-trained victims (%)	70.0	77.0	50.0	84.0 (Non-loss rate)	90.0

state-of-art approach. We also notice that masking almost has no impact upon our attack in the Sumo-Ants game, which further confirms that our adversarial agent exploits the game unfairness in this game. In addition, our attack establishes lower adversarial winning rate drop than the existing attack on the Starcraft II games. This confirms that our attack has a stronger exploitability than the existing attack on this sophisticated game. This may also indicate that our attack fails a victim via a stronger policy rather than triggering adversarial observations. Last but not least, we notice that masking even increases the adversarial winning rate of the existing attack on Sumo-Humans and Sumo-Ants.In [4], Gleave *et al.* also has the similar observation in their experiments. We suspect this is caused by the specific game rules of these two games. In future work, we will take a more closer look into the game rules together with the agent behaviors and find out the reasons behind this result.

Second, we collect the activations of the victim policy when playing with three different opponents: itself (a regular agent), our adversarial agent, and the adversarial agent obtained by the existing attack. We then follow the setup in [4] and use GMM and t-sne to demonstrate the differences among these sets of activations. Regarding GMM, we train two models with the activations collected from self-playing and playing against the existing attack. Then, we test these two models with these three sets of activations. The results are shown in Fig. S5 and Fig. S8. As we can first observe from these figures, on MuJoCo games, the results are aligned with our observations from the demo videos. That is, except for Sumo-Ants, our method exploits the similar weakness in victim policies with the existing attack. Regarding the StacrCraft II game, Fig. S5 and Fig. S8 shows that the victim agent demonstrates substantially different behaviors when playing against our adversarial agent and that obtained by the existing attack. Together with the attack performance in Fig. 1 and Table S3, these results indicate our attack obtains an adversarial policy that exploits different and more threatening weakness than the existing attack on this sophisticated game.

Attacking a mediocre victim. As is shown in Supplementary S5.2, the victim agents used in our experiments are well-trained agents. Here, we show that our attack could also demonstrate its effectiveness even if we train our adversarial agents against mediocre agents. Specifically, We first train a mediocre agent on each game by running fewer self-play iterations. The average winning rate of these agents against the well-trained victims is 18%, confirming their mediocre performances. Then, we train our adversarial agent against these agents and test it against the mediocre victim and the well-trained victim. Table S4 shows the attack performances. We observe that our attack is effective against mediocre & well-trained victims even if the adversary is pitched on mediocre.

Attacking a victim that varies its policy. In our evaluation, we fix the victim agents. Here, we conduct an initial exploration of our attack's effectiveness against a victim that varies its policy. Specifically, we train our attack with a victim that encodes two well-trained self-play policies and plays each one with an equal probability in each game round. Fig S6 shows the attack performance. The result confirms our attack's effectiveness against this dynamic victim. Our future works will test more non-fixed victims.

Table S5: The adversarial agent's non-loss rate against the victim agent and another regular agent.

		Kick And Defend	You Shall Not Pass	Sumo Humans	Sumo Ants	StarCraft II
Our attack	Victim (%)	91.0	89.0	94.0	89.0	98.0
	Regular agent (%)	70.0	60.0	82.0	88.0	92.0
Baseline attack	Victim (%)	83.0	84.0	93.0	61.0	64.0
	Regular agent (%)	60.0	64.0	74.0	45.0	58.0

S7 Attack Transferability

We also compare the transferability of our attack with that of the baseline attack [4]. Specifically, given a game, we first take a regular agent released in [2]. This agent is different from the on used for adversarial policy training. Then, we set up the regular agent to play with the adversarial agent learned through our method and that learned through the baseline approach [4]. In this experiment, we set each adversarial agent to play with the regular agent for 100 rounds and report the adversarial agent's winning rate. We compare the adversary's winning rate with the winning rate observed when the adversary plays with the victim it trains against. Through this comparison, we can measure the exploitability variation after transferring an adversarial agent to attack a different target agent. Recall that for each game, we randomly select six initial states and thus obtain six adversarial agents. In this experiment, we report the transferability of the strongest adversarial agent. Table S5 shows the transferability of our attack and that of the baseline attack [4]. First, we observe that both methods establish a certain level of transferability on the five games. Compared with the baseline attack, our attack demonstrates a slightly better transferability. We believe this is because of the stronger exploitability of our attack. As is also shown in the table, our adversarial policy in Sumo-Ants establishes the highest transferability. As is mentioned above, this policy exploits the game unfairness rather than the weakness of a specific victim policy. As such, it is less relevant to the opponent policy and thus has a stronger transferability. On the contrary, as for the adversarial policy that disturbs the victim observation via its action, its performance will be jeopardized when transferred to a different regular policy.

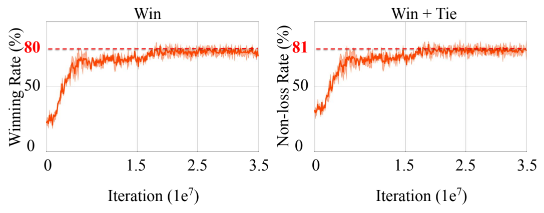


Figure S6: Our attack performance against a dynamic victim in the Kick And Defend game.

S8 Our attack vs. baseline in zero-sum setting

In Section 4, we show the advantages of our attack over the baseline [4] on real-world nonzero-sum games. Here, we demonstrate the performance of both methods in a zero-sum setting. Specifically, we first modify the reward design of the StarCraft II and make it a zero-sum game. [2] To achieve this, we remove all the intermediate rewards and only preserve the rewards related to the game results, i.e., 1 (win), 0 (tie), and -1 (lose). We then apply both our attack and the baseline to train an adversarial agent under the modified reward design and record the adversarial winning rate every time its policy is updated. Figure S7 shows the results of six runs. As we can observe from the figure, the adversarial agent trained by the baseline [4] demonstrates a similar winning rate with our attack. In zero-sum games, maximizing adversarial reward will automatically minimize

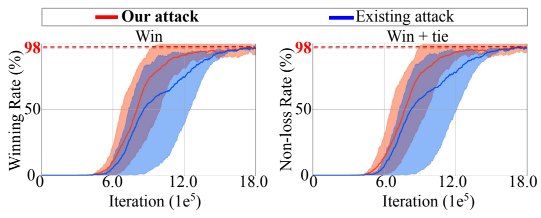


Figure S7: Performance comparison in a zero-sum game.

[2]The MuJoCo games cannot be transformed into zero-sum settings, because the agent is not able to pick up basic behaviors, such as standing and running, without the intermediate rewards.

the victim reward. As such, our approach's adversarial agent will demonstrate the same performance as that learned by the baseline [4] in this setting. It should be noted that the adversarial agent trained by the baseline in the zero-sum setting performs better than that in the nonzero-sum setting (Fig. 1). As is discussed above, this is because the state-of-art attack is less effective in nonzero-sum settings. It should also be noted that our learning converges much faster in the nonzero-sum setting (Fig. 1: 0.8M iterations) than it in the zero-sum setting (1.6M), which demonstrates the benefit brought by the intermediate rewards. In most of the two-player games, game developers design intermediate rewards to help RL agent training.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *Proc. of OSDI*, 2016.

[2] Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent complexity via multi-agent competition. In *Proc. of ICLR*, 2018.

[3] DeepMind. Alphastar: Mastering the real-time strategy game starcraft ii. `https://en.wikipedia.org/wiki/AlphaStar_(software)`, 2017.

[4] Adam Gleave, Michael Dennis, Neel Kant, Cody Wild, Sergey Levine, and Stuart Russell. Adversarial policies: Attacking deep reinforcement learning. In *Proc. of ICLR*, 2020.

[5] David A Levin and Yuval Peres. *Markov chains and mixing times.* American Mathematical Society., 2017.

[6] Ruo-Ze Liu, Haifeng Guo, Xiaozhong Ji, Yang Yu, Zhen-Jia Pang, Zitai Xiao, Yuzhou Wu, and Tong Lu. Efficient reinforcement learning with a thought-game for starcraft. *arXiv preprint arXiv:1903.00715*, 2019.

[7] openai. Stable baselines. `https://stable-baselines.readthedocs.io/en/master/`, 2018.

[8] Jan Peters, Katharina Mulling, and Yasemin Altun. Relative entropy policy search. In *Proc. of AAAI*, 2010.

[9] Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 2008.

[10] David Pollard. Asymptopia: an exposition of statistical asymptotic theory. 2000. *URL http://www. stat. yale. edu/pollard/Books/Asymptopia*, 2000.

[11] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proc. of ICML*, 2015.

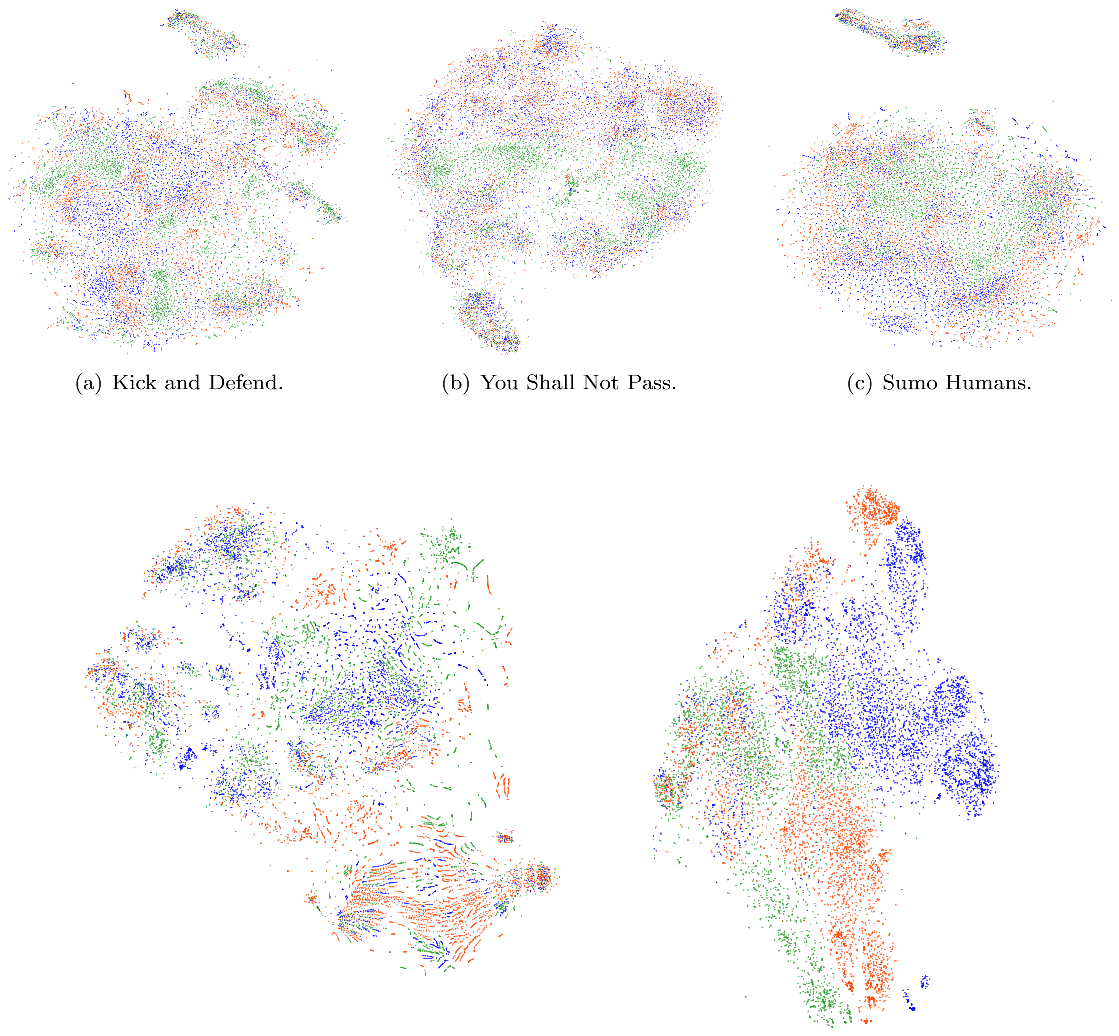
[12] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[13] Peng Sun, Xinghai Sun, Lei Han, Jiechao Xiong, Qing Wang, Bo Li, Yang Zheng, Ji Liu, Yongsheng Liu, Han Liu, et al. Tstarbots: Defeating the cheating level builtin ai in starcraft ii in the full game. *arXiv preprint arXiv:1809.07193*, 2018.

[14] Sebastian Thrun. Monte carlo pomdps. In *Proc. of NeurIPS*, 2000.

[15] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Proc. of ICIRS*, 2012.

[16] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.



(a) Kick and Defend.

(b) You Shall Not Pass.

(c) Sumo Humans.

(d) Sumo Ants.

(e) StarCraft II.

Figure S8: t-SNE visualizations of the victim activations when playing against different opponents in MuJoCo games. The green dots are the victim activations when setting a regular agent as the opponent. The red and blue dots indicates the victim activations when playing against our adversarial agent and that obtained by the existing attack, respectively.