RL-Course 2024/25: Final Project Report

Learning Through Reinforcement: Nils Großepieper, Mátyás Pólya

February 26, 2025

1 Introduction

In our final project of the Reinforcement Learning course, we were tasked with solving a custom gymnasium environment [6] that resembles a laser hockey game.. The actions include controlling our agent's x and y movement, turning, and shooting the puck if it is in the agent's possession. The observations are 18-dimensional vectors, which include information about the two players' and the puck's state. The following 4 values are also provided that can be used to create custom rewards for the agent: the winner of the episode, puck closeness, whether the puck is touched, and a value based on the puck's x velocity direction. Weak and strong opponents are also given for training and baseline purposes. The environment alternates the puck's position every episode, so the players are both attacking and defending. There are also two alternative modes where the agent only does one of the two.
In this report, we will present our solutions for solving this environment using different reinforcement learning algorithms.

- **Twin Delayed Deep Deterministic Policy Gradient** (Nils Großepieper)
- **Soft Actor-Critic** (Mátyás Pólya)

2 Twin Delayed Deep Deterministic Policy Gradient

The Twin Delayed Deep Deterministic Policy Gradient (TD3) [2] was developed to overcome the overestimation bias and stabilize policy learning, which are problems of the Deep Deterministic Policy Gradient (DDPG) algorithms. TD3 improves stability by using three main techniques: (1) twin Q-learning, which mitigates overestimation bias by maintaining two Q-functions and using the smaller value for target estimation, (2) target policy smoothing, which adds noise to the target action to reduce variance, and (3) delayed policy updates, which ensure the policy network is updated less frequently than the critics, leading to more stable training updates.

2.1 Methods

The TD3 algorithm consists of one policy network (actor) $\pi_\phi(s)$, two Q networks (critics) $Q_{\theta_1}(s,a)$ and $Q_{\theta_2}(s,a)$, along the target networks for each of the online networks $Q_{\theta'_1}(s,a)$, $Q_{\theta'_2}(s,a)$, and $\pi'_\phi(s)$.
Before training, the agent collects experience by interacting with the environment. Starting in state s, the policy-network $\pi_\phi(s)$ selects an action a. Based on the state s and the action a we receive a reward from the environment r and a transition to the new state s'. These experiences (s,a,r,s') are stored in the replay buffer D, from which the agent samples during training.
To improve exploration, we can add exploration noise ϵ to the predicted action of the policy network $\pi_\phi(s)$:

$$a_t = \pi_\phi(s_t) + \epsilon_e, \quad \epsilon_e \sim \mathcal{N}(0, \sigma) \tag{1}$$

After adding noise, the action is clipped within valid bounds to ensure feasibility.
During interaction with the environment, the agent stores experience tuples (s, a, r, s') in a replay buffer D. The replay buffer allows the algorithm to learn from past experience. When training starts, two objective functions are used. The first is used to update the Q-networks, and the second is used to update the Policy-network.

Each Q-network $Q_{\theta_i'}(s, a)$ is trained by minimizing the mean squared error (MSE) between the predicted Q-value and the target Q-value. The target value is calculated using the Bellman equation:

$$J_Q(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim D}\left[\frac{1}{2}\left(Q_{\theta_i}(s, a) - y(r, s', a')\right)^2\right] \tag{2}$$

Unlike in the traditional DDPG approach, the original paper [2] use the current reward r and the discounted Q-value estimations of both the target Q-networks for the state s' and the target action a'. This ensures that the critic learns conservative Q-value estimates, reducing the risk of overly optimistic policy updates. Only the smaller target Q-value is chosen to calculate the expected return. This approach prevents an overestimation bias:

$$y(r, s', a') = r + \gamma \min_{i=1,2} Q_{\theta_i'}(s', a') \tag{3}$$

We estimate the Q-values of the two target Q-networks based on the next state s' and the estimated target action a' of the target Policy-network $\pi_\phi'(s')$. To improve stability and prevent over-fitting to deterministic actions, policy noise is introduced when computing the target action a':

$$a' = \pi_{\phi'}(s') + \epsilon_p, \quad \epsilon_p \sim \mathcal{N}(0, \sigma), \quad \epsilon_p = \text{clip}(\epsilon_p, -c, c) \tag{4}$$

Unlike the Q-networks, policy network $\pi_\phi(s)$ is not updated every step to improve stability and performance. The objective is to maximize the Q-values under the policy:

$$J_\pi(\phi) = \mathbb{E}_{s\sim D}\left[Q_{\theta_1}(s, \pi_\phi(s))\right] \tag{5}$$

We use the first Q-network $Q_{\theta_1}(s, a)$ with state s and action a for the gradient update of the policy network $\pi_\phi(s)$.

$$\nabla_\phi J_\pi = \mathbb{E}_{s\sim D}\left[\nabla_a Q_{\theta_1}(s, a)\big|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)\right] \tag{6}$$

After updating the policy network $\pi_\phi(s)$ and the Q-networks $Q_{\theta_1}(s, a)$ and $Q_{\theta_2}(s, a)$, the target networks are updated using soft update rule to stabilize training with the hyperparameter τ:

$$\theta_i' \leftarrow \tau\theta_i + (1 - \tau)\theta_i', \quad \phi' \leftarrow \tau\phi + (1 - \tau)\phi' \tag{7}$$

2.2 Experiments

In our experiments, we chose to analyze the win rate over the last 50 games played against the weaker opponent.

2.2.1 Reward

We start by investigating the reward structure since it is one of the most influential parameter for the performance of our TD3 model. An episode can either end after 250 steps or when a goal has been scored. The agent receives in the "Standard Reward" case -10, 0, or 10 at the end of the episode based on whether it lost, draw, or won. In addition, the agent receives a penalty for each step the puck remains on its side of the field, with the penalty based on the distance between the agent and the puck. We chose three alternative reward structures to test if they improve the performance of the agent.
The first alternative reward structure "Dense Reward" adds two parameters. The first one rewards the agent for touching the puck and the second one rewards the agent for shooting the puck in the opponent's direction. For the second alternative reward structure, 'Dense Reward + Penalty,' we add a penalty for every time step the agent has not touched the puck yet when the puck is on the agent's own side of the field. We use the reward structure from "Dense Reward" in our fourth experiment "High Win Bonus" but instead of a reward of +10 for win we choose +20.

$$R(s,a) = \alpha R_{\text{result}} + \beta R_{\text{distance}} + \gamma R_{\text{touch}} + \delta R_{\text{shoot}} + \eta R_{\text{penalty}}$$

$$R_{\text{result}} = \begin{cases} +10, & \text{if agent wins} \\ 0, & \text{if draw} \\ -10, & \text{if agent loses} \end{cases}$$

where $\alpha, \beta, \gamma, \delta, \eta$ are weighting factors to balance each term.

In all experiments, $\alpha, \beta, \gamma, \delta \in \{0, 1\}$ and $\eta \in \{0, -0.001\}$.

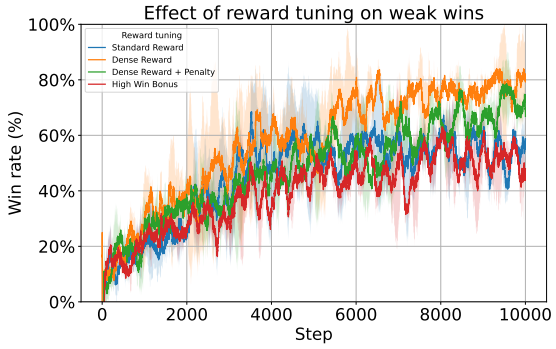

Figure 1: The effect of different reward structures on the win rate. The lines indicate the means over the different runs, while the shaded regions show the standard deviations.

As seen in Figure 1, the "Standard Reward" might have to few parameters for the agent to learn the game efficiently. While the reward structure of "High Win Bonus" might encourage winning, the excessive reward of +20 could destabilize training by inflating Q-values, increasing the risk of exploding gradients and overestimation bias, leading to inefficient policy updates. The denser reward functions, "Dense Reward" and "Dense Reward + Penalty", significantly improve learning efficiency by providing continuous feedback, helping the agent adjust its actions at each step rather than relying solely on sparse, delayed rewards at the end of an episode.

2.2.2 Exploration Noise

In the second experiment we replace the constant normally distributed exploration noise $\epsilon_e \sim \mathcal{N}(0, 0.1)$ with an exploration noise that decays exponentially over the episodes. It has been show that that decaying exploration noise improves the agent's outcome in various fields [1] [5]:

$$\sigma_e = \epsilon_0 \cdot \lambda^{-t} \tag{8}$$

$$a_t = \pi_\phi(s_t) + \epsilon_e, \quad \epsilon_e \sim \mathcal{N}(0, \sigma_e) \tag{9}$$

This approach ensures broad early exploration with a smooth exploration noise decay. This secures refined movement in later episodes with less random actions. The agent's performance with the constant standard exploration noise $\epsilon_e \sim \mathcal{N}(0, 0.1)$ has been compared to the performance of agents using exponentially decaying exploration noise, with varying initial noise level ϵ_0 and decay rates λ.

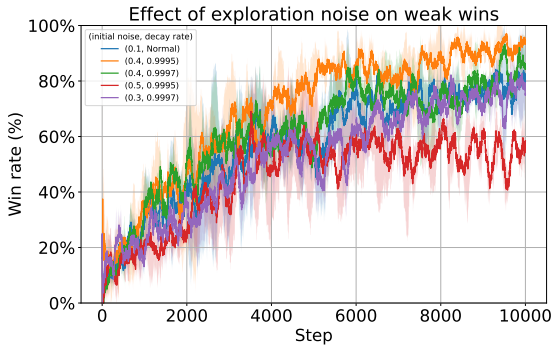

Figure 2: The effect of different exploration noises on the win rate. The lines indicate the means over the different runs, while the shaded regions show the standard deviations.

Almost all agents with an exponentially decaying exploration noise demonstrated improved performance, as seen in Figure 2, except for the run with an initial noise level of 0.5. The excessive initial randomness likely caused unstable and erratic training, leading to inaccurate Q-value estimations [5]. Exponentially decaying exploration noise can enhance the agent's outcome by enabling broader exploration in the early stages of training while gradually shifting towards more refined and stable decision-making as learning progresses.

2.2.3 Policy Noise

In the third experiment, exponential decay is applied to the policy noise:

$$\sigma_= \epsilon_0 \cdot \lambda^{-t} \tag{10}$$

$$a' = \pi_{\theta'}(s') + \epsilon_p, \quad \epsilon_p \sim \mathcal{N}(0, \sigma), \quad \epsilon_p = \text{clip}(\epsilon_p, -c, c) \tag{11}$$

This approach helps prevent overestimation bias in Q-values during early training while allowing for more precise action selection in later episodes. To analyze its effect, the performance of the standard constant policy noise ϵ_p is compared with different exponentially decaying noise settings. Testing exponential decay on policy noise provides insights into how gradually reducing randomness impacts policy stability and Q-value estimation. A well-adjusted decay rate can balance exploration and exploitation,

ensuring robust learning while preventing excessive action perturbations in later training stages. We avoid high-amplitude noise since TD3 can handle moderate noise levels better [7].

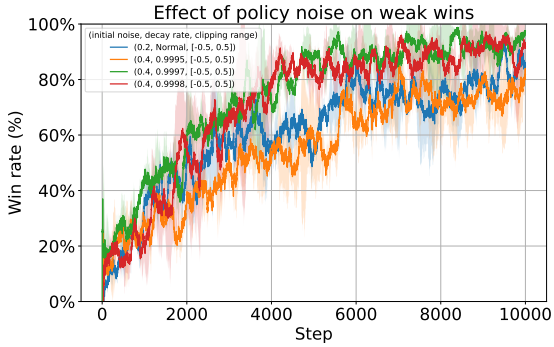

Figure 3: The effect of different policy noises on the win rate. The lines indicate the means over the different runs, while the shaded regions show the standard deviations.

The results in Figure 3 show that exponentially decaying policy noise can improve the agent's win rate by balancing exploration and stability. However, a decay rate that is too high leads to premature convergence, while a decay rate that is too low maintains excessive randomness, hindering strategy refinement.

2.2.4 Policy Delay

At last we test three different policy delay values 1, 2, and 3 which influence how often the Policy-network is updated.

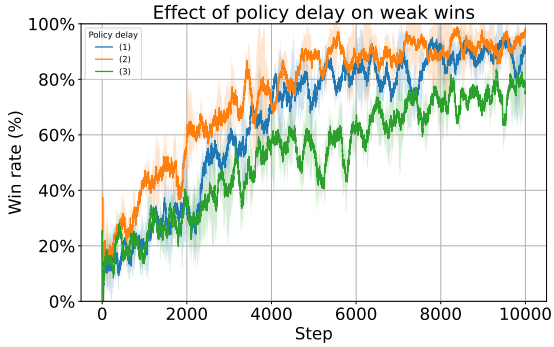

Figure 4: The effect of different policy delays on the win rate. The lines indicate the means over the different runs, while the shaded regions show the standard deviations.

As seen in Figure 4 policy delay of 2 yields the best results. A delay of 1 leads to excessive policy updates, causing unstable Q-values, while a delay of 3 results in slower policy adaptation, making the policy lag behind the critic updates [2].

3 Soft Actor-Critic

Soft Actor-Critic [3] was created to solve two problems: the sample inefficiency of on-policy algorithms, and brittle convergence properties of reinforcement learning methods. It tries to solve the exploration-exploitation problem by changing the reward term to include the entropy of the distribution of possible

next actions. This encourages the agent to explore instead of committing to an action. A result of this formulation is that if two actions are thought to be equally good, then they are assigned equal probabilities.

3.1 Methods

The objective of the policy is

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \left[r(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}(\pi(\cdot \mid \mathbf{s}_t)) \right] \tag{12}$$

The temperature parameter (α) controls the stochasticity of the optimal policy. The original objective can be recovered in the limit as $\alpha \to 0$.
The original paper [3] introduces and proves the correct convergence of soft policy iteration, which is used for the tabular case.

$$\mathcal{T}^\pi Q(\mathbf{s}_t, \mathbf{a}_t) \triangleq r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim p} \left[V(\mathbf{s}_{t+1}) \right], \tag{13}$$

where

$$V(\mathbf{s}_t) = \mathbb{E}_{\mathbf{a}_t \sim \pi} \left[Q(\mathbf{s}_t, \mathbf{a}_t) - \log \pi(\mathbf{a}_t \mid \mathbf{s}_t) \right]. \tag{14}$$

The Soft Actor-Critic architecture extends this method for non-tabular cases using function approximators. It has two Q-networks (θ_1, θ_2), a policy network (ϕ) and a value network with a target network $(\psi, \bar{\psi})$.
The objective function of the value network is

$$J_V(\psi) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[\frac{1}{2} \left(V_\psi(\mathbf{s}_t) - \mathbb{E}_{\mathbf{a}_t \sim \pi_\phi} \left[Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - \log \pi_\phi(\mathbf{a}_t \mid \mathbf{s}_t) \right] \right)^2 \right]. \tag{15}$$

The objective function for the Q-networks is

$$J_Q(\theta_i) = \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_{\theta_i}(\mathbf{s}_t, \mathbf{a}_t) - \left(r(s_t, a_t) + \gamma \, \mathbb{E}_{s_{t+1} \sim p} \left[V_{\bar{\psi}}(s_{t+1}) \right] \right) \right)^2 \right]. \tag{16}$$

The objective function for the policy network is

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[D_{KL} \left(\pi_\phi(\cdot \mid \mathbf{s}_t) \middle\| \frac{\exp(Q_\theta(\mathbf{s}_t, \cdot))}{Z_\theta(\mathbf{s}_t)} \right) \right], \tag{17}$$

which can be rewritten to

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}, \epsilon_t \sim \nu} \left[\log \pi_\phi(f_\phi(\epsilon_t; \mathbf{s}_t) \mid \mathbf{s}_t) - Q_\theta(\mathbf{s}_t, f_\phi(\epsilon_t; s_t)) \right], \tag{18}$$

using the reparametrization trick, in the case where the output of the policy network is a normal distribution.
The minimum of the two Q-functions is used in Equation 15 and Equation 18 to prevent overly optimistic values.
In the follow-up paper [4] the authors introduced a method for learning the α parameter with the following objective function:

$$J(\alpha) = \mathbb{E}_{\mathbf{a}_t \sim \pi_t} \left[-\alpha \log \pi_t(\mathbf{a}_t | \mathbf{s}_t) - \alpha \bar{\mathcal{H}} \right], \tag{19}$$

where $\bar{\mathcal{H}} = -\dim(\mathcal{A})$. The paper also abandons the explicit value function, and instead uses Equation 14 implicitly with the Q-functions.

3.2 Experiments

In our experiments, we chose to analyze the win rate over the last 50 games played against the weaker opponent.

3.2.1 The entropy parameter

Both papers [3] [4] highlight that α is the most important hyperparameter that influences the success of the algorithm. We tested three cases: keeping it constant throughout the training, annealing it exponentially, and using the adaptive learning method in Equation 19.

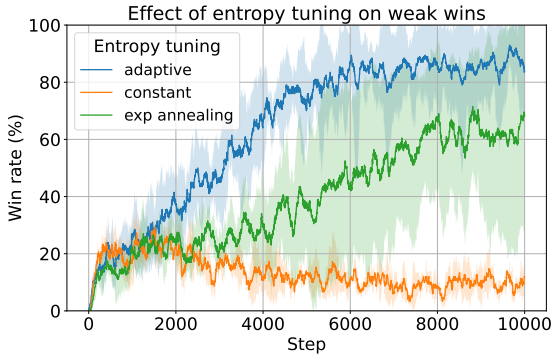

Figure 5: The effect of entropy tuning on the win rate. The lines indicate the means over the different runs, while the shaded regions show the standard deviations.

As we can see in Figure 5, the adaptive tuning of the α parameter performed the best out of the three methods. Keeping the entropy parameter constant performed the worst, which is expected since the agent never transitions from the exploration phase to the exploitation phase.

3.2.2 Reward

An episode ends if a goal is scored, or after approximately 250 steps. In the original reward formulation, the agent receives a reward of -10, 0, or 10 at the end of the episode, based on whether it lost, there was no goal, or it won the episode. For each steps that the puck is in the agent's side of the court, the agent also receives a penalty based on the distance between the agent and the puck.
We wanted to examine the effects of augmenting this reward by multiplying the *puck closeness* reward by different numbers, but hypothesized that if the multiplier were too high, the agent would sometimes prefer to let the puck into its goal to prematurely end the episode, so it wouldn't receive the high penalties. To prevent this, we introduced a way of penalizing this behavior: starting from the beginning of the episode, at every step, the agent receives a high penalty if it has not yet touched the puck, but when it touches the puck for the first time, it receives a reward that negates all the penalties received up until that point.
We also tried using only the rewards corresponding to the outcome of the episode, but were unsuccessful because of the sparsity of the rewards.
In Figure 6 we can see that the inclusion of the *puck not touched yet* penalty had a positive effect on the win rate, with a more noticeable improvement on runs where the *puck closeness* multiplier was high. For the *puck closeness* reward, the original formulation outperformed the modified versions.

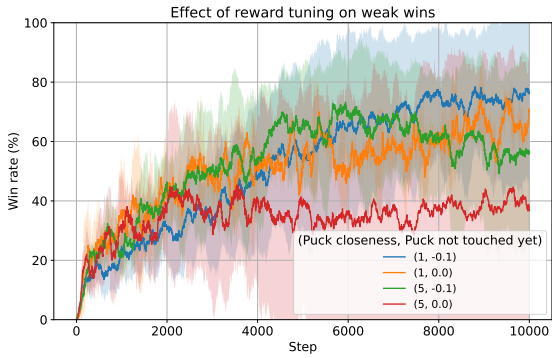

Figure 6: The effect of different reward formulations on the win rate. The lines indicate the means over the different runs, while the shaded regions show the standard deviations.

3.2.3 Soft vs. hard target update

During training, some agents had noticeably higher Q-function losses compared to others, which we attributed to an unfortunate combination of the learning rate, soft target updates, and changes in the entropy parameter. In the SAC architecture, modifying the entropy parameter α also implicitly changes the correct Q-function values.
The SAC paper [3] also reported that some environments benefit from hard target updates instead of soft target updates, so we decided to compare the two.

$$\bar{\theta}_i \leftarrow (1-\tau)\bar{\theta}_i + \tau\theta_i \tag{20}$$

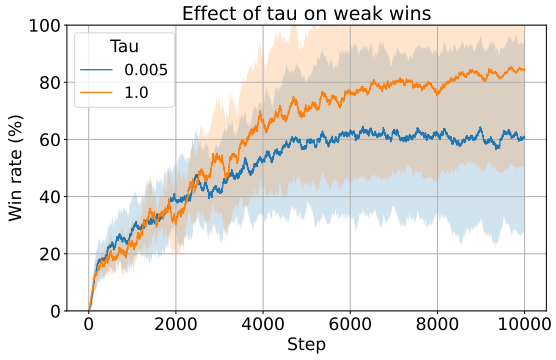

Figure 7: The effect of using different kind of target updates on the win rate. The lines indicate the means over the different runs, while the shaded regions show the standard deviations.

As seen in Figure 7, using hard target update ($\tau = 1$) benefited the performance of the agents. It also solved the problem with the high Q-function losses.

4 Discussion

We chose to examine the 3 best models in regard of their win rate against the weak opponent. In Figure 8 we can see that the SAC models achieved higher performance compared to the TD3 models. This aligns with our observations; however, we must note that more experiments were conducted on the SAC model,

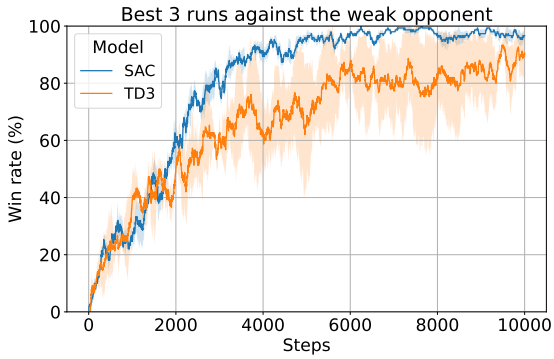

Figure 8: The win rate of the best 3 models against the weak opponent. The lines indicate the means over the different runs, while the shaded regions show the standard deviations.

which may have contributed to an inflated value. The SAC model also exhibits lower variance, a fact that was also reported in the paper [3]. The better performance of SAC can also be attributed to its better sample efficiency, better handling of stochastic environments (we don't know the opponent's action), and more robustness to hyperparameter tuning. The final performance of the two models are similar, as each model brings its own strength and weaknesses.

The project can be accessed on GitHub at: `https://github.com/Matyi00/rl-hockey-homework.git`

References

[1] A. Aghanim, H. Chekenbah, O. Oulhaj, and R. Lasri. Q-learning empowered cavity filter tuning with epsilon decay strategy. *Progress In Electromagnetics Research C*, 140:31–40, 2024.

[2] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596. PMLR, 10–15 Jul 2018.

[3] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *ArXiv*, abs/1801.01290, 2018.

[4] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine. Soft actor-critic algorithms and applications. *ArXiv*, abs/1812.05905, 2018.

[5] J. Hollenstein, S. Auddy, M. Saveriano, E. Renaudo, and J. Piater. Action noise in off-policy deep reinforcement learning: Impact on exploration and performance. *Transactions on Machine Learning Research*, November 2022.

[6] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. D. Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, K. Arjun, R. Perez-Vicente, A. Pierr'e, S. Schulhoff, J. J. Tai, H. Tan, and O. G. Younis. Gymnasium: A standard interface for reinforcement learning environments. *ArXiv*, abs/2407.17032, 2024.

[7] A. Xu. Robustness exploration of the twin-delayed deep deterministic policy gradient algorithm under noise attack. In *Proceedings of the 2023 International Conference on Data Science, Advanced Algorithm and Intelligent Computing (DAI 2023)*, volume 180 of *Advances in Intelligent Systems Research*. Atlantis Press, 2024.