Comparing the Effectiveness of PPO and its Variants in Training AI to Play Game

Luobin Cui[1], *IEEE,* and Ying Tang[2], *Senior Member, IEEE*

2023 International Conference on Cyber-Physical Social Intelligence (ICCSI) | 979-8-3503-1249-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICCSI58851.2023.10303944

***Abstract*—Automated game intelligence is a crucial step in rapid game development. A promising research direction for automated game intelligence is reinforcement learning, and specifically, t he p roximal p olicy o ptimization (PPO) algorithm. Two variants of the PPO, Maskable PPO and Recurrent PPO, further extend the capabilities of the PPO. We compare the performance of these three algorithms in the 2D game Mario and a 3D car racing game environment. We also evaluate their performance and applicability by comparing the experimental results of the original algorithm authors. With our results, we provide recommendations on PPO configuration d epending on the target game type, providing future developers with a benchmark to help them decide which algorithm is most applicable for their applications.**

***KEYWORDS*—Game AI agents, deep learning, Proximal Policy Optimization**

I. Introduction

Game artificial i ntelligence (AI) a nd g ame t esting p lay a crucial role in game development. In the early days, Developers needed to manually write rules and algorithms to define decisions and behaviors of game AIs. As game evolved, the complexity of game AI increases, so does the development process that requires more time and effort. Game testing faces a similar problem, where testing efforts significantly increase as games become more complex.

The rise of machine learning and deep learning in recent years has brought a dramatic change to games development and testings [1]. By using deep neural networks and reinforcement learning, AI agents can learn from data and automatically optimize their behaviors [2]. This approach not only makes game AI smarter but also speeds up the game development process with reduced workload.

While these techniques are straightforward to build, they might not fit all types of games as games vary in characteristics and genes.

In action-adventure games, players complete level challenges by controlling the game character to perform actions such as jumping, running, and attacking. Thus, the type of game AI with a strong ability to react quickly and navigate on these moves is desirable. Similarly, strategy games would like to build the game, AI with long-term planning and decision-making capabilities to develop effective strategies and respond to player actions. To achieve speed and victory in racing games, game AI must be able to drive and navigate with the track perception [3], Therefore, it is important and challenging in game development to choose the most suitable machine learning technique for game AI with respect to game types and genres.

To tackle this challenge, this paper explores on one of widely used algorithms for game AI–Proximal Policy Optimization(PPO)- with the focus on two recent variants, Maskable PPO and Recurrent PPO, Although the superiority of Maskable PPO and Recurrent PPO over the traditional PPO has been evaluated since their inception, these tests are limited. For instance, Maskable PPO was only applied to a strategy game [4]. Thus, its applicability to other types of games is unknown. A similar situation applies to Recurrent PPO, where it was only tested through a series of mini-games. With this consideration, this paper fills this gap, making the following contribution:

- An empirical study is designed to analyze the performance of Maskable PPO and Recurrent PPO in two types of games: action and 3D racing games. We used Mario for the action game.
- In addition to the strength of Maskable PPO and Recurrent PPO, their limitations are explored. Give tasks in action games lack of timing dependency, it is difficult for them to take good advantages of Recurrent PPO. For the same reason, Recurrent PPO outperforms Maskable PPO in racing games.
- Our findings provide a valuable reference for the choice of game AI

The rest of the paper is organized as follows. Section 2 briefly introduces the three PPO algorithms; Section 3 discusses our experiment set-up; Section 4 presents the evaluation results, followed by our conclusion.

II. Methodology

A. Proximal Policy Optimization

The proximal policy optimization algorithm (PPO) is a reinforcement learning algorithm proposed by Schulman et al., 2017 [5]. The PPO algorithm is a deep reinforcement learning algorithm based on policy gradients [6] that avoids large policy updates that may harm performance by limiting the deviation of new policies from old ones. The Trust Region Policy Optimization (TRPO) [7] algorithm is a predecessor of the PPO algorithm and uses a stricter restriction to control

This work was supported in part by the National Science Foundation under Grant 1913809 and the U.S. Environmental Protection Agency under Grant 84034701

[1]Luobin Cui is with the Department of Electrical and Computer Engineering, Rowan University, Glassboro, NJ 08028 USA. Email: cuiluo77@students.rowan.edu.

[2]Ying Tang is with the Department of Electrical and Computer Engineering at Rowan University, Glassboro, NJ 08028, USA; and School of Information Science and Technology, Dalian Maritime University, Dalian, China. Email: tang@rowan.edu (the corresponding author).

the magnitude of policy updates to ensure the stability of learning. However, TRPO is difficult to implement and scale, especially for discrete action spaces or auxiliary losses [8]. To overcome these drawbacks, the PPO algorithm proposes new objective functions and implements small batches of updates in multiple training steps. First, the agent executes the current strategy and uses the collected data to estimate a value function in the strategy evaluation step. This value function is used to calculate the dominance estimate, which represents the advantage of an action over other actions in a given state. Then, The strategy parameters are updated to maximize the expected payoff. To ensure that the strategy update is not too drastic, PPO introduces an alternative objective function that combines the dominance of the new strategy with the dominance of the old strategy. This approach aims to balance the improvement of the new strategy with maintaining stability. The objective function of PPO can be written as Eq. (1).

$$L^{PPO}(\theta) = \hat{\mathbb{E}}_t\Big[\min\Big(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t\Big)\Big] \tag{1}$$

θ represents the parameter associated with the policy network. The variable t denotes the specific time step under consideration. The function A_t represents the advantage function at time step t.$r_t(\theta)$ denotes the likelihood ratio between the new policy and the old policy at time step t, as defined by Eq. (2).

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{2}$$

$\pi_\theta(a_t|s_t)$ represents the probability of selecting action a_t in state s_t according to the new policy.

The objective function is designed to balance two objectives: maximizing the expected reward and preventing the policy from changing too much at once. The first term in the min function encourages the policy to move in the direction of higher advantage, while the second term restricts the size of the policy update by clipping the likelihood ratio. The parameter ϵ controls the degree of clipping.

In this paper, specifically, we focus on improved versions of the PPO algorithm.

B. Maskable Proximal Policy Optimization

Maskable PPO is an improved version proposed by Huang et al.,2020 in [4]. This algorithm improves the performance of the PPO for continuous action control tasks by using MAF to approximate the action probability distribution in the continuous action space.

The objective function of the Maskable PPO is similar to that of the PPO, but the mask parameter m is introduced in the construction of the policy network and the value network. Specifically, the output of the strategy network is $\pi_\theta(a|s,m)$, where θ denotes the parameters of the strategy network, a denotes the action, s denotes the state, and m denotes the mask parameter. The output of the value network is $V_\phi(s,m)$, where ϕ denotes the parameters of the value network. the objective function of the Maskable PPO can be expressed as Eq. (3). (4).

$$L(\theta,\phi) = \mathbb{E}_t\left[\min\left(X * A_t^{\text{Clip}}, X\right) - c_1 V_\phi Y + c_2\mathcal{H}(\pi_\theta Y)\right] \tag{3}$$

$$X = \frac{\pi_\theta(a_t|s_t,m_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t,m_t)}, Y = (s_t,m_t) \tag{4}$$

where A_t^{Clip} is the pruned dominance function, c_1 and c_2 are hyperparameters, and $\mathcal{H}(\pi_\theta(s_t,m_t))$ is the entropy of the strategy $\pi_\theta(s_t,m_t)$. By introducing the mask parameter m, the Maskable PPO can share part of the neural network structure between different tasks.

C. Recurrent Proximal Policy Optimization

The Recurrent PPO [9] is based on a variant of the PPO, whose objective function is similar to that of the PPO, but uses recurrent neural networks (RNNs) in the neural network structure to process sequential data. Specifically, Recurrent PPO uses RNN models such as LSTM to process data in continuous state space and uses RNN to record past game states during strategy updates to enhance the temporal and memory capabilities of the strategy.

The key advantage of the Recurrent PPO is that by using RNNs, it can better handle the "temporal relationship and continuity of game states, and can learn from past game states to improve the generalization ability of the model. An additional loss term is added to the objective function of the Recurrent PPO to optimize the memory ability and sequence prediction ability of RNNs. This gives it better performance in handling reinforcement learning problems in continuous state space.

III. Experiments

The purpose of this study is to compare the performance of the PPO, Recurrent PPO, and Maskable PPO in training intelligent agents for playing the NES Mario game [10], and a 3D strategy racing game. The racing game is made in the Unity game engine. We also used ML-Agents Gym Wrapper [11] to create the game training environment. The algorithms are all from Stable-Baselines3 [12].

For the Mario game we conducted 12 experiments, in which the standard and maskable PPO algorithms were trained using two different policy networks, CNN Policy, and MLP Policy, while the Recurrent PPO was trained using CNNLSTM policy and MLPLSTM policy. The reason for the division into two neural network structures, CNN and MLP, is that they each have their advantages for different types of data and tasks. CNN can accept two-dimensional or multidimensional shapes of input data, and image data. CNN can efficiently process local features in images using convolutional operations while pooling layers can extract the spatial hierarchy of features. MLP usually spreads the input data into one-dimensional vectors and feeds them directly to the fully connected layer for processing. The CNNLSTM policy and MLPLSTM policy are based on the original one with the addition of long short-term memory networks.

In the default parameter experiments, we kept the algorithm default parameters fixed and trained each of the three algorithms using different policy networks. To further optimize algorithm performance, we conducted hyperparameter optimization experiments. In the optimization process, we trained each set of parameters for 200,000 steps and tried 100 different parameter sets. We used the Bayesian optimization algorithm to find the best set of parameters. [13]

Each experiment ran for 10 million steps, and we focused on evaluating the following three parameters:

- rollout/ep_rew_mean: Average reward is an important indicator of the performance of a reinforcement learning algorithm. In this experiment, this parameter will show the performance of each algorithm in terms of average reward at the end of each epoch, helping to evaluate the efficiency of the algorithm.
- train/entropy_loss: Entropy loss refers to the entropy value of the current policy function, which is a balance between determinism and randomness. The smaller the entropy loss, the more deterministic the policy is, and the larger the entropy loss, the more random the policy is. In this experiment, this parameter will show the balance between randomness and determinism in the algorithm.
- train/loss: Training loss is an important indicator of model training performance. This parameter will show the training loss performance of the algorithm in each epoch. In this experiment, the training loss performance will reflect the efficiency and stability of the algorithm during the learning process.

In CNN Policy, we preprocessed the original game image, converting it to grayscale, and overlaying it four times as input. In MLP Policy, we read the original game image from memory and processed it into state shape.

In the Racing game shown in Fig. 11, the player takes the role of hemoglobin and drives the car through the blood vessels. In the game, each level generates random elements such as oxygen, lipids and cholesterol on the track. The player's goal is to transport oxygen and complete the task of hemoglobin in the blood vessels. Bonus points are earned by colliding with oxygen, which is obtained while reducing acceleration and maximum speed. The hemoglobin zone will cause the player to slip and make maneuvering more difficult. Cholesterol will move around the track and if it collides with the player, it will slow down the player's forward speed, requiring the player to pay attention to avoidance. The game also sets a time limit, players need to complete the level within the time limit while trying to get the highest possible score.

We conduct 6 experiments and will applied the MLP policy throughout to train the entire game. For performance comparison, we choose a certain number of steps to test the game, and we use 500 million (5e10) steps of the game as a benchmark for comparison. By comparing different algorithms, we were able to evaluate the performance of the AI in the game and determine which algorithm was able to reach the point in the most efficient way and achieve the highest score within the specified number of steps. We are able to better observe and analyze the differences in performance of different algorithms when faced with specific tasks and constraints.

Fig. 1. Mario Mean Reward

IV. Experimental Results and Analysis

A. Results

We trained Mario using three algorithms: PPO, Maskable PPO, and Recurrent PPO and compared the performance of each algorithm under four different parameter settings, including the optimal and default parameters for CNN Policy and MLP Policy.

We calculated the mean average reward of each algorithm to evaluate the performance of the model. The experimental results are shown in Fig.2.

In the picture, M stands for Maskable PPO, R stands for Recurrent PPO, Opt stands for Preferred Parameters, and Def stands for Default Parameters.

We can see that the performance of the Recurrent PPO in the Mario game is poor, with all four curves relatively low. This indicates that this algorithm is not suitable for this type of game.

The Recurrent PPO is based on the recurrent neural network (RNN) PPO, mainly used to deal with continuous action space problems. The characteristic of RNN is that it can capture the state and relationship of time series data, making it suitable for tasks that require consideration of time series relationships. However, in the Mario game, the state at each moment is determined only by the current screen image and player actions, without a strong time-series relationship. Moreover, the state and action space in the Mario game are relatively small and do not require complex models to handle. Therefore, the Recurrent PPO may perform poorly when dealing with this type of game, as it may introduce unnecessary time series relationships into the model, increasing computational burden and noise.

In addition, the Recurrent PPO also needs to deal with longer time series data, which may lead to a more complex and time-consuming training process, thus affecting the training effect. The poor performance of the Recurrent PPO in the Mario game may be due to its design features and algorithm complexity being unsuitable for this type of game.

As shown in Fig.3, the performance of the default parameters for PPO_CNN Policy and Maskable PPO_CNN Policy is also poor. These results indicate that the performance of the CNN Policy may not meet expectations in the Mario game and that the default parameter settings cannot meet the needs

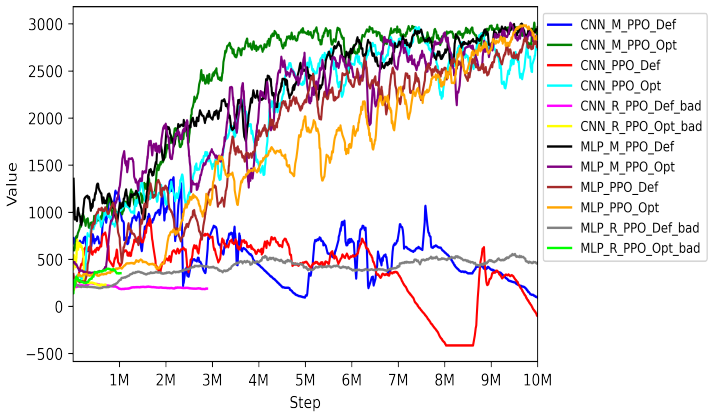

Fig. 2. Mario Mean Reward

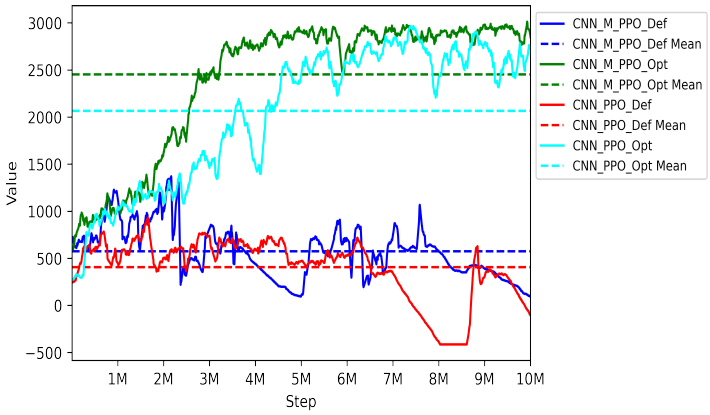

Fig. 3. Mario CNN Policy Mean Reward

of the game. The CNN Policy is usually used to process visual information, such as image or video data. In the Mario game, the input for each state is a screen image, and the screen image of the Mario game is relatively small and does not change dramatically. Therefore, in this case, using CNN for feature extraction and processing may be too complex, leading to a decrease in model performance. Generally, better results were obtained after optimizing the parameters for MLP Policy.

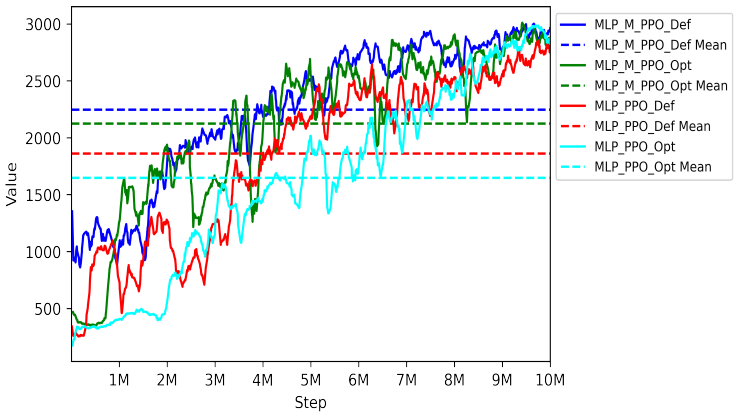

Fig. 4. Mario MLP Policy Mean Reward

As shown in Fig.4, we compared the performance of the MLP Policy policy with default parameters and optimized parameters after 10 million steps. Surprisingly, the optimized parameters performed worse than the default parameters in terms of average score rate, but outperformed the default parameters in terms of the highest score. This result may be due to the optimized parameters causing the model to focus more on achieving the highest score, at the expense of the average score. It's worth noting that the highest score may be a relatively difficult goal to achieve in the game, while the average score better reflects the model's overall performance throughout the game. The optimized parameters also performed better than the default parameters in terms of value loss, as shown in Fig.6. This indicates that the optimized parameter model is better able to accurately predict the state value function, thereby improving model stability and training effectiveness.

Optimized parameters are likely to focus more on long-term results. To improve the performance of the agent, it is usually necessary to balance the trade-off between exploration and exploitation. Under default parameters, the MLP Policy policy may be more inclined to exploitation to achieve faster score rates, but after long-term training, overfitting may occur, resulting in the agent being unable to adapt well to new states. Under optimized parameters, the MLP Policy policy may focus more on exploration and learning new strategies, thus achieving better long-term performance. This also explains why the optimized parameters perform better in terms of the highest score because they are better able to explore the environment and find better strategies. In order to better

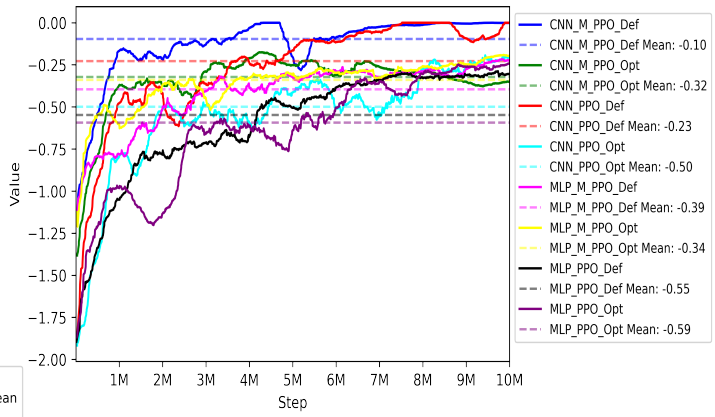

Fig. 5. Mario Entropy Loss

observe the results, both Fig.5. and Fig.6. have been smoothed. Based on Fig.5., it can be seen that in each algorithm, as the number of training steps increases, the entropy loss gradually decreases and stabilizes at a low value. This indicates that as the model learns and optimizes, the model's confidence in the current policy increases, thereby reducing the need for exploratory behavior, i.e., the entropy of exploratory behavior decreases. In both CNN Policy and MLP Policy, the entropy loss under optimized parameters is lower than that under default parameters, indicating that the model under optimized

parameters is more confident and has less exploratory behavior, making the model more stable.

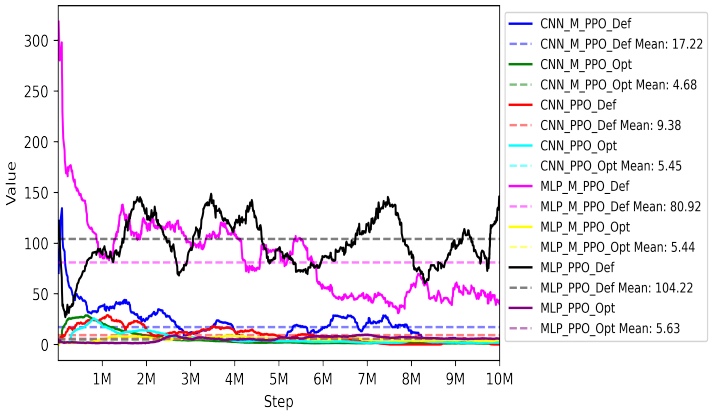

Fig. 6. Mario Value Loss

In the 3d strategy racing game, we will focus mainly on the mean reward. As shown in Fig.7. With parameter defaults, both improved algorithms outperformed the original PPO, suggesting that they can be trained and optimized more efficiently in some respects. However, when hyperparameter preferences were performed, surprisingly, Maskable PPO did not outperform the original PPO. This may imply that in some environments or tasks, the masking operation does not yield significant performance gains. This finding emphasizes the need to take into account the variability of different environments and tasks when designing improved algorithms.

On the other hand, Recurrent PPO performs well with both parameter settings and achieves the highest scores early on. This suggests that the use of recurrent neural networks can better capture sequence information in tasks with long-term dependencies, which is more advantageous in games with simple scene repetition, thus improving the performance of the algorithm.

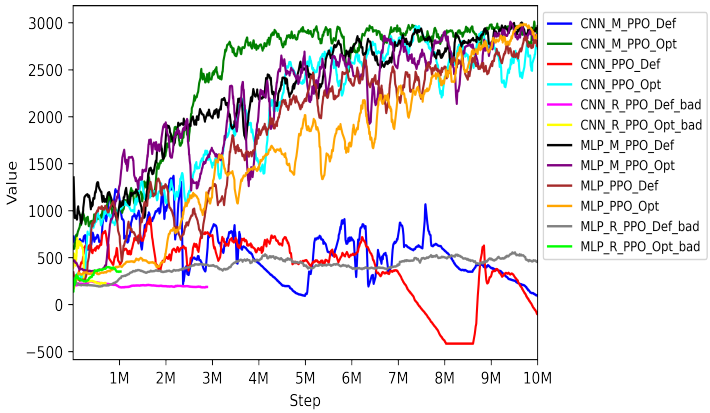

Fig. 7. 3d Strategy Racing Mean Reward

B. Analysis

The experimental results show that Recurrent PPO performs poorly in the Mario game, but excels in the racing game. In contrast, Maskable PPO outperformed PPO when training the Mario game with complex operations, but did not show better performance than regular PPO in the relatively simple racing game.

The original authors of Maskable PPO compared Maskable PPO and PPO in microRTS (a strategy game) and found that Maskable PPO showed excellent performance in that game. Although the two game types, game mechanics and perspectives are not identical, they share similarities in operational complexity and redundant information. Therefore, when training game intelligence, choosing Maskable PPO can speed up the progress of game training when faced with game content redundancy.

The original authors of Recurrent PPO compared Recurrent PPO with PPO in MuJoCo (a simulated physics environment for things like robot walking and robot arms moving objects) and CarRacing (a top-down racing game). again, Recurrent PPO showed excellent performance. These games all share a feature of individual game intelligence learning, involving both physics simulation and driving behavior. In the 3D strategy racing game tested, the learning actions through the history of the intelligence contributed to the subsequent driving performance. It can be seen that LSTM policies have better performance in learning skill games.

Moreover, it is observed that the PPO with default parameters hardly produces good results under the CNN policy. However, by optimizing the parameters, the performance of the PPO significantly improves and even exceeds the scoring rate of the MLP policy. This indicates that the parameters have an important impact on the performance of the CNN strategy.

V. Conclusion

This paper compares the performance of three reinforcement learning algorithms, namely PPO, Maskable PPO, and Recurrent PPO, in training the 2D game ”Mario” and the 3D strategy racing game. Maskable PPO performs well in both games and outperforms the original algorithm. Recurrent PPO performs better in the racing game than Maskable PPO, while it performs poorly in the Mario game and fails to capture long-term dependencies. The appropriate choice of PPO should be based on the operational complexity and scene complexity of the game. Given that Recurrent PPO consumes a significant amount of memory, it is recommended that Maskable PPO be preferred to achieve prompt training results. This study provides valuable data and insights for algorithm selection in the field of game AI development, as well as game testing. We believe that these algorithms will find broader applications in the future and hope that this research will contribute to further advancements in this field.

References

[1] P. Migdał, B. Olechno, and B. Podgórski, ”Level generation and style enhancement – deep learning for game development overview,” *arXiv*, 2107.07397, 2021. [Online]. Available: https://arxiv.org/abs/2107.07397.

[2] A. C. Roibu, ”Design of Artificial Intelligence Agents for Games using Deep Reinforcement Learning,” *arXiv*, 1905.04127, 2019. [Online]. Available: https://arxiv.org/abs/1905.04127.

[3] S. Lecchi, ”Artificial intelligence in racing games,” in *2009 IEEE Symposium on Computational Intelligence and Games*, 2009, pp. 1-1. doi: 10.1109/CIG.2009.5286512.

[4] S. Huang and S. Ontañón, "A Closer Look at Invalid Action Masking in Policy Gradient Algorithms," *CoRR*, vol. abs/2006.14171, 2020. [Online]. Available: https://arxiv.org/abs/2006.14171.

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy Optimization Algorithms," in *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[6] A. Ilyas, L. Engstrom, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, "A Closer Look at Deep Policy Gradients," *arXiv*, 1811.02553, 2020. [Online]. Available: https://arxiv.org/abs/1811.02553.

[7] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. "Trust Region policy Optimization." *CoRR*, vol. abs/1502.05477, 2015. Available: http://arxiv.org/abs/1502.05477

[8] F. Otto, P. Becker, N. A. Vien, H. C. Ziesche, and G. Neumann, "Differentiable Trust Region Layers for Deep Reinforcement Learning," *arXiv*, 2101.09207, 2021. [Online]. Available: https://arxiv.org/abs/2101.09207.

[9] M. Pleines, M. Pallasch, F. Zimmer, and M. Preuss, "Generalization, Mayhems and Limits in Recurrent Proximal Policy Optimization," *arXiv*, 2205.11104, 2022. [Online]. Available: https://arxiv.org/abs/2205.11104.

[10] C. Kauten, "Super Mario Bros for OpenAI Gym," [Online]. Available: https://github.com/Kautenja/gym-super-mario-bros, 2018.

[11] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents," *arXiv preprint arXiv:1809.02627*, 2020. [Online]. Available: https://arxiv.org/pdf/1809.02627.pdf.

[12] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1-8, 2021.

[13] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.