You Teach Me and I Teach You: Reinforcement Learners in Pokémon Showdown

OpenAI Gym Leaders:
Jeremy Chen, Lee Ding, Chace Hayhurst, George Hu, Wasu (Top) Piriyakulkij

December 16, 2021

Abstract: Pokémon is one of the most popular video game series in the world. A primary component of Pokémon games is the turn-based Pokémon battle, which presents an interesting space for researchers to study reinforcement learning systems. We developed a TD-Lambda Actor-Critic framework with novel entropy regularization that learns to play the Pokémon game via the Pokémon Showdown online simulator. We show that there are promising results in learning Pokémon battles with fixed teams, especially under constrained conditions.

1 Introduction

The Pokémon series of role-playing games is one of the best-selling video game franchises in the world. At the heart of Pokémon is the Pokémon battle, which pits two players against each other in a team battling system. Both players have the same number of Pokémon, but each Pokémon has different moves. Players must adaptively choose playing strategies based on the opponent's Pokémon team and also on the opponent's playing strategy. Pokémon battles are turn-based, partially-observable, and stochastic, which are not usually present in many current game-based reinforcement learning settings. The unique properties of Pokémon battles, combined with the game's ubiquity, present a prime setting for studying reinforcement learning systems.

Here, we applied a number of reinforcement learning-based approaches for playing Pokémon battles. In particular, we experimented with an A2C [6] TD-Lambda-based model trained against various hand-designed Pokémon battling algorithms. Furthermore, we demonstrated how a self-play mechanism can be used to boost our performance. We then evaluated our models through several different battle configurations in Pokémon Showdown, an online open-source Pokémon battle simulator. In constrained scenarios where we limited factors such as randomness and the number of Pokémon per party, our models were able to learn reasonably effective policies. However, as we loosened battle constraints, we found that playing Pokémon battles soon became too complicated of a problem for our applications of standard reinforcement learning algorithms to learn.

2 Background

2.1 Pokémon Battles

A standard Pokémon battle is a turn-based game involving two players. Typically, Pokémon battles involve a human player battling an AI player. However, two human players can also compete against each other. Players begin a battle with a party of up to six Pokémon, with one active Pokémon each.

Each Pokémon is allocated a certain number of hit points (HP), which represents the total amount of damage a Pokémon can receive before fainting (being removed from gameplay during the battle). In addition, each Pokémon possesses a unique set of moves that can reduce the opposing Pokémon's HP or boost its abilities for the next turn (such as increasing damage dealt). During each turn in the battle, both players simultaneously choose moves for their active Pokémon. Players have some, but not all, information about the abilities and moves of the opposing Pokémon throughout the battle, which means that a Pokémon battle is a partially-observable game. In addition, Pokémon battles are stochastic because the effectiveness of a Pokémon's move varies randomly and some moves have a chance to miss. The goal of the game is to reduce the HP of all the opponent's Pokémon to 0 while avoiding this for your Pokémon. Hence, Pokémon battles are zero-sum games.

To study reinforcement learning frameworks for playing Pokémon battles, we use Pokémon Showdown, an open-source web-based Pokémon battle simulator. Pokémon Showdown supports Single Battles for all Pokémon generations as well as Double Battles for Generation V-VIII Pokémon. We interact with Pokémon Showdown for agent training and evaluation using Poke-env, an API that allows us to manipulate Pokémon Showdown-related objects. For example, we can use Poke-env to configure Pokémon roster sizes and the specific Pokémon in those rosters. Poke-env also exposes an OpenAI Gym interface that enables us to train and evaluate agents under different settings in Pokémon Showdown. The main Pokémon Showdown configurations that we study are battles with random and custom Pokémon rosters of varying lengths. Full documentation on the Poke-env API can be found at https://poke-env.readthedocs.io/en/latest/.

2.2 Reinforcement Learning for Pokémon Battles

We can use Poke-env to represent the state space in a Pokémon Showdown battle as a vector containing information about the player, their opponent, and the current game environment. The dimensionality of the Pokémon Showdown state space is equal to 339, which is quite high. Hence, when studying reinforcement learning approaches in this space, we would need to consider the tradeoff between the state size and the agent's ability to converge.

Along with the state space's high dimensionality, there are other challenges that we need to account for when training reinforcement learning Pokémon Showdown agents. First, creating an effective Pokémon battle-playing agent typically requires the creator to have a large amount of domain knowledge about the Pokémon game. Another challenge is managing the complexities that arise during a battle. Agents also need to deal with a high degree of stochasticity with their moves. Broadly speaking, there is only a small set of best decisions that an agent can take in any given situation during a battle.

2.3 Previous Work

In 2017, Lee and Togelius [4] proposed Pokémon battles as an interesting challenge in the Reinforcement Learning domain because they are partially observable, have a high degree of stochasticity, and involve rewards that are possibly delayed over a very long period. Since then, there has been much work on training reinforcement learning agents to play in Pokémon battles, particularly deep learning methods to represent the Q function because of the game's rather high dimensional state space. For example, Simões et al. [7] implemented the WPLθ and GIGAθ with ϵ-greedy exploration deep competitive reinforcement learning algorithms to generate agents for constrained Pokémon battle scenarios. In such scenarios, the authors found that both algorithms were able to successfully converge to effective strategies, such as tactically swapping active Pokémon for type advantage.

Huang and Lee [2] were able to train an actor-critic model to play Pokémon through self-play. The agent was able to outperform `pmariglia`, a state-of-the-art search-based method for playing Pokémon that involves its own evaluation heuristic. This paper is possibly most applicable to our work because the authors used the same random team selection that we train our agents to play with.

Our work builds upon these previous approaches to create intelligent agents that are capable of playing simulated Pokémon battles in Pokémon Showdown.

3 Problem Statement: Pokémon Battle Formulation

We represent a Pokémon battle as a Partially Observable Game with the formulation (S, A, T, R, γ). An important note is that this formulation is relatively incomplete since encapsulating all the information in a Pokémon battle within the standard MDP format significantly increases the complexity.

- S is our state space, which we will elaborate on later.
- A is our action space, which is an index from 0 to 9.
  - 0 to $a_m - 1$ represent the moves the current active Pokémon can use. $a_m \leq 3$ is the number of available moves.
  - 4 to $3 + a_s$ represent the possible other Pokémon the current active Pokémon can switch to. $a_s \leq 6$ is the number of available other Pokémon to switch to. (In reality $a_s \leq 5$, but Poke-env forces 6).
- T is the transition function. This is stochastic and determined by the Pokémon game mechanics which we explained earlier. An important facet is that even though a Pokémon battle is adversarial, we frame T so that $T(s_k, a_k, s_{k+1})$ is the transition probability from the beginning of turn k to the beginning of turn $k + 1$ for the same player; the opponent's decision is implicitly placed here.
  - We are doing model-free RL, so this is not provided to our model.
- R is the reward function. The general objective is solely to win the match, so we assign $R(s, a) = 2$ where (s, a) leads to a game win and $R(s, a) = -2$ for a game loss. To aid in training, we shape the rewards as such:
  - Gain/lose 1 for having the opponent's active or your active Pokémon faint, respectively.
  - Gain/lose f for dealing f fraction of damage or healing f fraction of damage.
  - Gain/lose 0.1 for the opponent's active or your active Pokémon having a negative status effect, respectively.

  We looked into other reward values, but this was the most common reward setup that we used.

3.1 State Representation

We represent our state S as a vector of dimensionality $n = 339$. The state vector encodes the following components of an ongoing Pokémon battle:

- **Current Active Pokémon Moves:** Base Power, Damage Multiplier, Accuracy, Heal Percentage, Move Category, Status Applied
- **Player/Opponent-Specific Information:** Current Types, Current Health, Stats, Current Boosts, Status Effects
- **Available Pokémon Switches:** Types, Current Health, Stats
- **Battle Environment:** Side Conditions (e.g. spikes), Fields, Weather
- **Previous Action:** (Zeroes if first move of the battle)

Categorical variables in this vector, such as typing, are one-hot encoded.

4 Methodology

4.1 Model Architecture

Our model architecture is inspired by A2C [6] (Advantage Actor-Critic Deep Learning).

- $f_a : \mathbb{R}^{|S|} \to [0,1]^{|A|}$ is the actor network such that for state s, $f_a(s) = \pi(s)$; the output is the agent's policy distribution of actions. f_a is a 3-layer fully connected MLP, and a mask + softmax at the end. The mask filters out invalid actions, and the softmax ensures output is a probability distribution. We represent the weights for the actor as ϕ_a.
- $f_c : \mathbb{R}^{|S|} \to \mathbb{R}$ is the critic network such that for state s, $f_c(s) = V(s)$; the output is the our agent's value estimate for the state s. f_c is a 3-layer fully connected MLP. We represent the weights for the critic as ϕ_c.

4.2 Loss

TD Error: In practice, we found that the advantage function $A(s,a,s') = \gamma V(s') + R(s,a) - V(s)$ from A2C failed to work well, so we employed an eligibility trace to emulate TD(λ) Actor-Critic, similar to [3]. Hence, during a training match for successive states and actions s_1, a_1, s_2, a_2, s_3, ..., we let

$$\delta(s_k, a_k, s_{k+1}) = \sum_{j=1}^{k} \gamma^{k-j} \lambda^{k-j} A(s_j, a_j, s_{j+1})$$

be the TD(λ) error at state s_k.

Actor Loss: As established by Mnih et al. [6], we use the standard policy gradient provided our TD(λ) error.

$$L_p(s,a,s') = -(\ln \pi(a \mid s))\delta(s,a,s')$$

But, as Matheron et al. [5] note, a common failure mode for actor-critic methods is having the actor network converge to an incorrect policy too quickly before the critic network can properly evaluate states. Entropy-based regularization methods, such as Soft Actor-Critic [1], have been proposed, and our novel formulation in a similar vein is as follows:

Let $M(s)$ be the size $|A|$ vector for the 1-0 mask of valid actions for state s. We construct the masked uniform distribution

$$U(s)_i = \frac{\mathbb{1}_{\{M(s)_i = 1\}}}{\sum_{j=1}^{|A|} M(s)_j}$$

Now, we construct an additional penalty $L_e(s)$ for having our policy distribution $\pi(s)$ differ too greatly from $U(s)$

$$L_e(s,a,s') = \frac{\beta_1}{\ln(|A|)}(1 + \beta_2 \text{ReLU}(-\delta(s,a,s')))D_{KL}(\pi(s), U(s))$$

where D_{KL} is the Kullback-Leibler Divergence, $\delta(s)$ is the TD(λ) error at state s, and β_1 and β_2 are parameters. Note how this is set up to incentivize exploration more when $\delta(s,a,s')$ is negative (meaning we are getting unexpected poor results). Our total actor loss is

$$L_a(s,a,s') = L_p(s,a,s') + L_e(s,a,s')$$

Critic Loss: Our critic loss is just a square error for the advantage function, but with a coefficient η so that we can scale the actor and critic loss differently.

$$L_c(s,a,s') = \eta(\gamma V(s') + R(s,a) - V(s))^2$$

4.3 Training Methodology

The literature on how to train a Pokémon Reinforcement Learning Agent is sparse. Simões et al. [7] and Huang et al. [2] both use self-play as their primary training mechanism, but our attempts with only using self-play for training were unsuccessful. Instead, we used self-play in combination with two heuristical algorithms provided in Poke-env to train the model:

1. **Max Damage Bot**: An agent that naively picks the move with the highest base power. Forced swaps are chosen randomly.
2. **Smart Heuristic Bot**: An agent that employs a sophisticated heuristic based upon the state of the match. The agent always chooses super-effective moves if available based upon typing, and it also swaps out of poor match-ups and boosts itself in very favorable match-ups. It sets entry hazards early in the match, and occasionally uses effective status moves like toxic or moves that debuff the opponent as appropriate.

Let $w_s, w_m, w_h \in \mathbb{N}$ be the weights for self-training, max damage, and heuristic. For each episode, we do the following k times:

- Run w_m matches versus max damage bot, w_h matches versus heuristic bot, and w_s matches using self-play (versus a copy of the bot from the beginning of the episode). For these matches, we sample from the distribution $\pi(s)$ from the actor network to choose moves.
- After each match, collect all the (s, a, s') pair, and then do a batch calculation to find the losses $L_a(s, a, s')$ and $L_c(s, a, s')$. For $\bar{L}$ representing the mean loss over all (s, a, s') pairs, we update

$$\phi_a -= \alpha \nabla_{\phi_a} \bar{L}_a(s, a, s') \qquad \phi_c -= \alpha \nabla_{\phi_c} \bar{L}_c(s, a, s')$$

4.4 Match Configurations

- **6v6 Stationary Teams**: Both agents have the same fixed team of 6 Pokémon, and the lead Pokémon is randomly chosen. We considered two types of teams:
  - A balanced team, with Pokémon that know mostly offensive moves with a few defensive moves. The general strategy is to either attack or swap out for the "glass cannon" Pokémon, and to apply status effects and stall with the defensive Pokémon.
  - A "wall" team, a specific type of team composition that looks to defeat the opponent using damage-over-time statuses and healing moves, along with Pokémon with a lot of health, to try to outlast the opponent's damage.
- **6v6 Random Opponent**: Our agent has the same fixed team of 6 Pokémon every round. However, the opponent team is randomly chosen and ordered from a small set of 22 Pokémon that contained the most potential Pokémon types and niches.
- **4v4 Stationary Teams**: Both agents have the same fixed team of 4 Pokémon. This situation should be easier for our agent to learn since we have reduced the complexity of the game itself.

5 Evaluation

We evaluated our agent at the end of each episode for $m = 50$ games each against the Max Damage and Heuristic Agents explained earlier, as well as against an agent that randomly chooses moves each turn. We choose actions during evaluation by doing $\text{argmax}_{i \in A} \pi(s)_i$.

5.1 Results

5.1.1 6v6 Stationary Teams

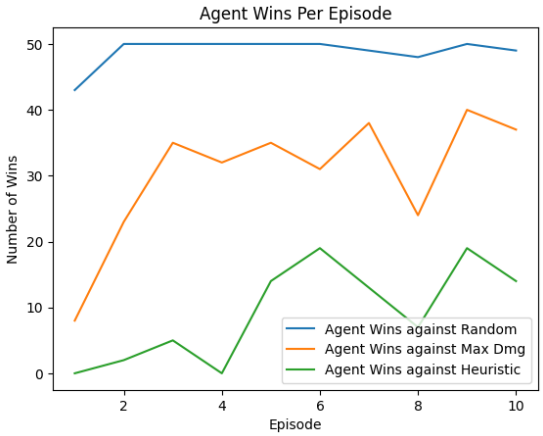

Figure 1: Number of wins out of 50 against Random, Max Damage, and Heuristic Bot.

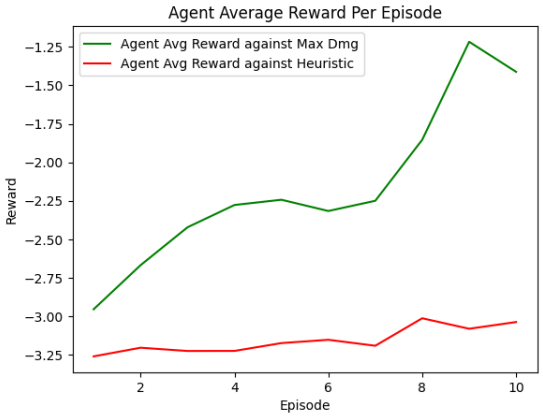

Figure 2: Average reward per turn for each episode against Max Damage and Heuristic Bot.

From these figures, we can see that the win rate of our agent has a general increasing trend against the three bots. Notice that the agent wins virtually all games against the random bot by the second episode. Additionally, our win rate and average reward per episode are much lower against the heuristic bot than any other opponent. Evidently, it is much more difficult for our agent to perform well against the heuristic. This is expected, as the heuristic incorporates domain knowledge about Pokémon whereas the other bots use very naive approaches.

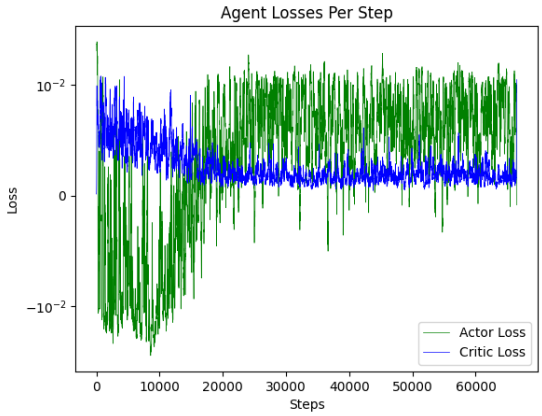

Figure 3: Number of wins out of 50 against Random, Max Damage, and Heuristic Bot.

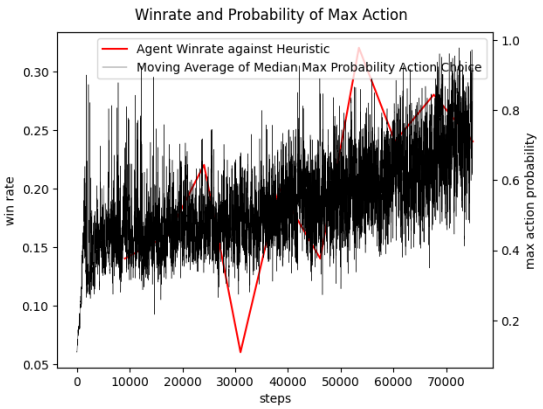

Figure 4: Average reward per turn for each episode against Max Damage and Heuristic Bot.

Since the policy loss is $L_p(s, a, s') = -(\ln \pi(a \mid s))\delta(s, a, s')$, we can can see that a negative actor loss means that $\delta(s, a, s') < 0$. Our value function estimate is failing to account for negative rewards, so the initial few tens of thousands of steps have our agent exploring the poor actions. Then for the rest of the training, the agent is mostly getting positive TD-error, meaning that we are slowly finding better than expected strategies and incorporating them. The critic loss (which is bounded below by zero) is steadily decreasing, meaning our critic is becoming more accurate. This is the desired behavior.

From figure 4, we can see that the median maximum probability for the policy distribution π is increasing slowly throughout. Earlier without the entropy regularization L_e, this would jump to 1 within 10000 steps and get stuck there. The entropy regularization ensures that the probabilities don't converge too fast before performance against the Heuristic has increased.

5.1.2 6v6 Random Opponent

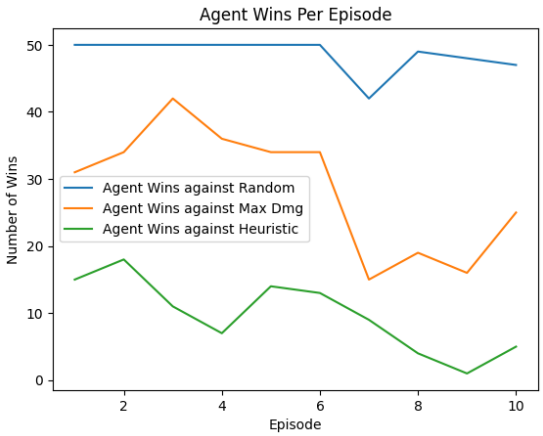

Figure 5: Number of wins out of 50 against Random, Max Damage, and Heuristic Bot.

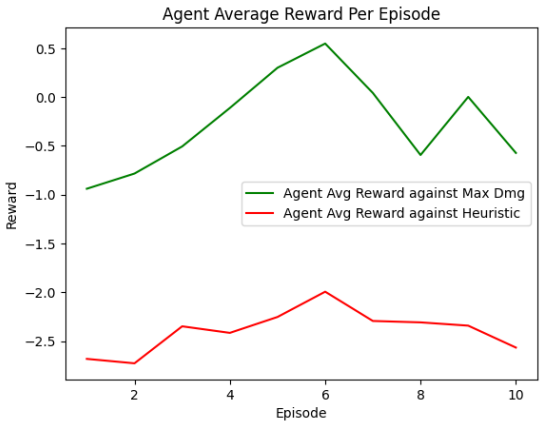

Figure 6: Average reward per turn for each episode against Max Damage and Heuristic Bot.

Interestingly, in the 6v6 case, our agent exhibits a downward trend as training extends past 5 episodes. In the beginning, the performance is comparable to the fully trained fixed 6v6 case, surprisingly, which could be because the hand-picked fixed composition is generally better than a random one. We hypothesize the decrease is due to the model overfitting to the teams it has seen before and generalizing poorly against other teams.

5.1.3 4v4 Stationary Teams

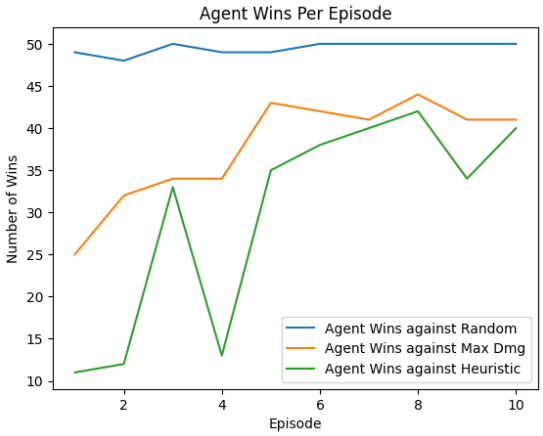

Figure 7: Number of wins out of 50 against Random, Max Damage, and Heuristic Bot.

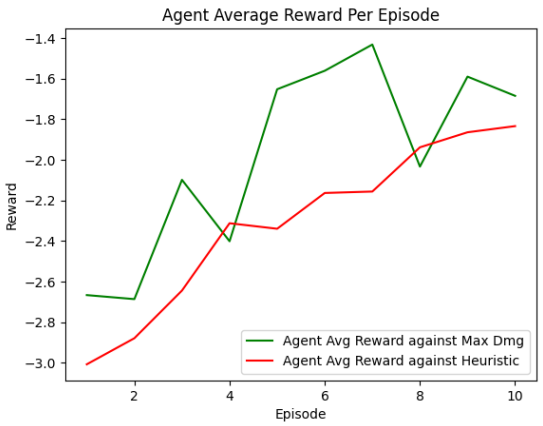

Figure 8: Average reward per turn for each episode against Max Damage and Heuristic Bot.

As we can see, win rates in the 4v4 case are much higher, particularly against the heuristic bot. This is a very large contrast to the 6v6 case where our agent was winning around 30-40% of games at the end of training. These results are expected because, as we discussed earlier, reducing the number of Pokémon reduces the complexity of the game making it much easier to learn.

6 Conclusion

Pokémon battles pose unique challenges not typically seen in game-based reinforcement learning settings. With the Pokémon Showdown simulator, we attempted to develop an A2C TD-Lambda-based model that can learn to play Pokémon battles. We experimented with different training approaches, including training against hand-designed Pokémon battling algorithms and incorporating self-play mechanisms. Our results highlight the sheer complexity of Pokémon battles as a reinforcement learning problem. Our bot was not able to compete with a hand-designed heuristic bot, even after extensive model tuning and training. However, when we enforced constraints on battle configurations, we performed quite adequately as our bot was able to beat two naive battle approaches. If we were to relax some of the simplifications that have been made from the original Pokémon games, this learning problem would quickly become much too complex for our agent to learn.

6.1 Next Steps

Our results also highlight three possible avenues for future work with reinforcement learning for Pokémon battles:

1. **Model Complexity**: one of the main issues that we considered was our model architecture. We believe it would be worthwhile to pursue different types of model architectures on this problem to see if another model can perform better in this task. One idea we had to potentially tackle this problem was the idea of using encoders to encode information about specific Pokémon and moves, which could help with cutting down on dimensionality and could help with more accurately representing battles.
2. **Generalization**: Past just the small set of Pokémon that was used, there are far more different types of Pokémon that can be encountered in a fully random Pokémon Showdown battle–how much more difficult does the problem get when expanding to the full set? In addition, many other different types of fixed team compositions could be tested–can a model successfully learn other types of strategies and teams, other than the two proposed here?
3. **State Representation**: as stated earlier, our state representation excludes specific features like held items simply due to adding lots of dimensions for probably very little benefit. If these features were incorporated, how much would the model improve? Is there a better way to embed all the information within a battle, to help with the information-computation time tradeoff?

References

[1] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[2] Dan Huang and Scott Lee. A self-play policy optimization approach to battling pokémon. In *2019 IEEE Conference on Games (CoG)*, pages 1–4, 2019. doi: 10.1109/CIG.2019.8848014.

[3] Hajime Kimura and Shigenobu Kobayashi. An analysis of actor/critic algorithms using eligibility traces: Reinforcement learning with imperfect value function. In *ICML*, 1998.

[4] Scott Lee and Julian Togelius. Showdown ai competition. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 191–198. IEEE, 2017.

[5] Guillaume Matheron, Nicolas Perrin, and Olivier Sigaud. Understanding failures of deterministic actor-critic with continuous action spaces and sparse rewards. In Igor Farkaš, Paolo Masulli, and Stefan Wermter, editors, *Artificial Neural Networks and Machine Learning – ICANN 2020*, pages 308–320, Cham, 2020. Springer International Publishing. ISBN 978-3-030-61616-8.

[6] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

[7] David Simões, Simão Reis, Nuno Lau, and Luís Paulo Reis. Competitive deep reinforcement learning over a pokémon battling simulator. In *2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 40–45, 2020. doi: 10.1109/ICARSC49921.2020.9096092.