# A Game Engine is all you need

Radu Ciobanu

West University of Timisoara

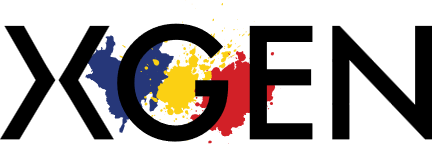
Faculty of Mathematics and Computer Science

Department: Computer Science

Email: radu.ciobanu02@-uvt.ro**Abstract**

*Reinforcement Learning (RL] stands as a fundamental approach for optimization problems, characterized by an iterative experimentation to maximize the cumulative reward of an agent performing in its environment. This paper highlights how game engines serve as the perfect training environments for intelligent agents, leveraging state-of-the-art RL algorithms.*

**Keywords:** *deep reinforcement learning, game engine*



## Introduction

Mapping inputs to outputs it’s not as easy when we do not have a target output that we are trying to learn, thus by deciding a reward function assembled as an environment that manages to offer an agent a general clue to the right improvement direction, we can estimate a solution by interfering more or less stochastically within that training environment. Tournament based methods between mutated policies depicted the naive incipient options, further expanded towards topology preservation as in NEAT algorithm [5], One does showing scalability capabilities, so the research left derivative free optimization (DFO) in exchange for algorithms that make use of differentiable neural networks as a better approach that still keeps the focus on the policy. Attempting to implement a system that does replicate either a simple supervised learning training or even a state-of-the-art deep reinforcement learning algorithm will end up with a limited knowledge overhead if there is no prior understanding or no deep learning framework usage. This paper covers in summary the development of a deep learning library from the ground up that can be found on GitHub[[1]](#footnote-0), and it inherits sections from [1], which covers in depth how neural networks work, train and integrate with Proximal Policy Optimization and Soft Actor-Critic.

Named **DeepUnity** and written for Unity in C Sharp programming language, it encompasses the common tools and modules present in any deep learning framework (with GPU acceleration support). Packaged within the repository there are RL environments examples with 3D modeled agents running by the means of Unity physics system, trained with DeepUnity using custom setups, personal modular utilities and experimental methods for optimization. While still in development at the moment this paper was written, a deep reinforcement learning interface in Godot utilizing TorchSharp can be found within another repository[[2]](#footnote-1).

## Preliminaries

**Reinforcement Learning** involves two key components: the agent and the environment. The agent, acting as a character to which is assigned a ”brain”, interacts within a world crafted by its creators—the environment. This world is purposefully designed, featuring obstacles and rewards to facilitate learning. Successfully solving a reinforcement learning problem occurs when the agent accumulates a predetermined amount of rewards or surpasses initial expectations within a defined timeframe — referred to as maximizing the expected reward. The agent’s interaction with the environment unfolds through transitions between states, triggered by specific actions. Each transition yields a reward, a numerical signal reflecting the quality of the action—a principle encapsulated in the Markov Decision Process (MDP). Temporal difference learning dictates that all preceding actions contributing to a reward are deemed favorable. Even after achieving success, the agent persists in exploring the environment, seeking to unveil concealed elements.

**Policy Gradient Methods** (PGM) represent one subclass in RL algorithms that directly optimizes a policy π parametrized by θ using gradient ascent. In any PGM method, the parameters of the policy are updated gradually towards the direction of the policy gradient, defined by the partial derivative of the expected cumulative reward with respect to each parameter θ. They are characterized by their ability to efficiently handle uncertainty and randomness in the environment. Unlike deterministic policies, which prescribe a single action for a given state, stochastic policies provide a probability distribution over the possible actions. This inherent stochasticity introduces a crucial element of exploration, enabling agents to discover more effective strategies in complex environments.

In the following, a brief introduction to two state-of-the-art algorithms will be presented, one on-policy and the other off-policy. A more in-depth explanation of both is regarded in [1], along with the gradient of the objective functions and implementation guidelines.

## Proximal Policy Optimization

PPO plays as the most efficient algorithm in terms of convergence speed and space complexity due to its on-policy nature. In the reference implementation, it was highly improved with several methods from [6], enumerating Early Stopping, Input Normalization, Reward Normalization, Generalized Advantage Estimate [4] and Advantage Normalization. The main loop of PPO can be shortly described as follows: collect a certain amount of trajectories, compute the target values, compute the advantages, minimize the Value network mean squared error and maximize the surrogate objective function.

The surrogate loss function is formulated to encourage an increase in the probability of actions that have positive advantage values (here denoted by A] and a decrease in the probability of actions with negative advantages. Its objective is to move the policy in the direction that improves performance by gradient ascent, within a limited step range by ceiling the probability ratio rt(θ) to 1 +ϵ. Mathematically speaking, the min and clip operators involved by the ceiling are not differentiable if their arguments are equal, though we are considering them differentiable by straight-through gradient estimate, similar to ReLU activation (which is not differentiable when x = 0).

L CLIP (θ) = E [min (r(θ)A , clip(r(θ), 1 − ϵ, 1 + ϵ)A)

The theory behind the differentiation of L CLIP and pseudocode of PPO can be found in [1], and the implementation at DeepUnity’s GitHub repository[[3]](#footnote-2).

## Soft Actor-Critic

In comparison to PPO, SAC requires a lot more parameter optimization in order to remain sample efficient, and since DeepUnity’s computation speed cannot fairly compare with strong tensor libraries like Torch or TensorFlow, PPO remains the preferred algorithm to be used.

SAC employs an extensive replay buffer to store all experiences, still used till the end of the training (in contrast to on-policy methods that clears the buffer post-update). The maximum-entropy characteristic in comparison to DDPG signifies SAC’s dual objective: it doesn’t only seek to maximize the expected cumulative rewards, but it also strives to maximize the entropy of the policy. SAC optimizes a trade-off between these objectives, introducing stochasticity to the agent’s behavior so it becomes less deterministic resulting in a more exploratory behaviour that doesn’t limit the convergence to a local optima. So the algorithm rewards the Q-values based not only on the received reward for a state-action pair, but also based on the stochasticity of the taken actions, and this is visible in the formula below that determines the Q networks targets:

Q\_target ϕ,t = r\_t +γ(1−d\_t)(min i=1,2Qϕtarg,i (s\_(t+1), a\_(t+1) −β log πθ(a\_t+1|s\_t+1), a\_t+1 ∼ πθ(·|s\_t+1)

The implementation uses the twin version of SAC described on *openai/spinningup*[[4]](#footnote-3) documentation, because the Q-function is slowly overestimating Q-values leading to disrupting the policy, and the problem was addressed by taking the minimum between two Q networks similar to Double Q-Learning and TD3.

The objective function, which is the loss of the policy network, is defined as the negative log-likelihood of an action a in a state s times an entropy coefficient β ∼ 0.2 added to the minimum of the Q values estimators given by (s, a\_θ(s)) pair:

L (θ) = min i=1,2 Q\_ϕ,i (s, aθ) − β log π\_θ(aθ|s)

The theory behind the differentiation of the objective function and pseudocode of SAC can be found in [1], and the implementation at DeepUnity’s GitHub repository[[5]](#footnote-4).

## Game engines and current technologies

The industry of game development increased considerably in the past years, as well as the tools needed to improve the production rate. Not only that the mainstream game engines like Unity and Unreal received a multitude of tools and cutting-edge technologies, but also the open-sourced alternatives gained popularity amongst indie developers since they provide enough resources to work with.

As the reference implementation framework suggests, it is integrated within Unity. For reinforcement learning, Unity puts on the table great possibilities when it comes to simulation, enumerating Unity’s Data-Oriented Technology Stack (DOTS) or MuJoCo physics system integration, but it lacks the integration of a deep learning library, since all tensor .NET NuGet libraries yield incompatibilities. Developed by Unity Technologies, ML Agents[[6]](#footnote-5) is an open-sourced Unity package that allows training RL agents, but with certain constraints. Godot, the most popular open-source game engine, allows .NET packages to run flawlessly and the community supports good physics alternatives (e.g. Jolt).

## Reference Implementation

DeepUnity is an add-on framework that provides tensor computation (with GPU acceleration support) and deep neural networks, along with reinforcement learning tools that enable training for intelligent agents within Unity environments using Proximal Policy Optimization (PPO) or Soft Actor-Critic (SAC). Due to its inherent limitations in general performance, which hinder its competitiveness against frameworks enjoying robust community engagement and operating without constraints, its implementation has been exclusively tailored to provide modest enhancements in efficiency, ease of use, and educational value for the purposes of this paper, devoid of any competitive orientation.

The deep learning library, beside metrics and contains the following:

1. Modules/Layers

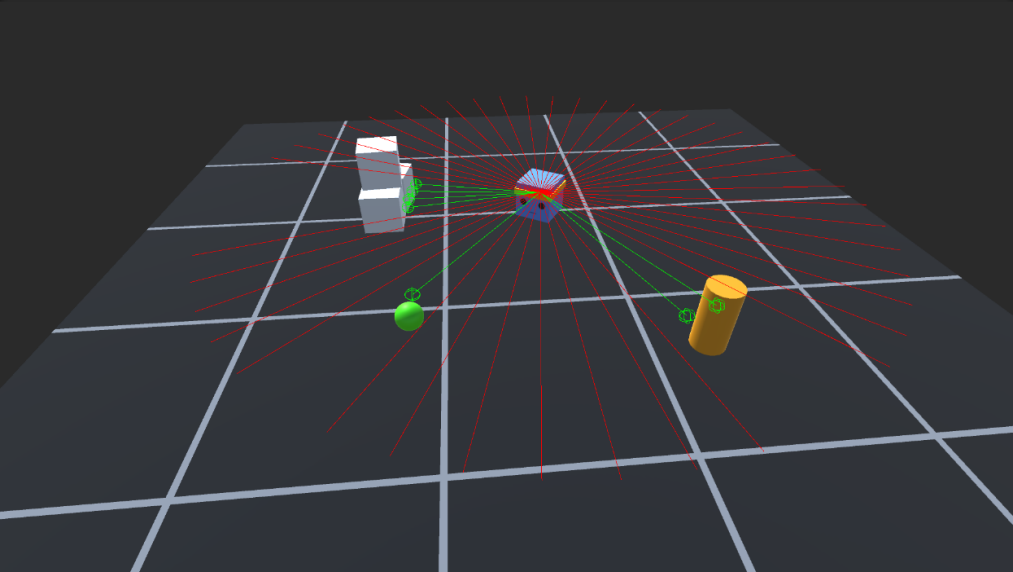
* **Dense**
* Nonlinear **Activations** (15+)
* **Conv2D**
* **RNNCell**
* **Batch**/**Layer**/**RMS Norm**alization
* **MultiheadAttention**
* **Dropout**
* Max/Avg **Pooling** & Zero/Mirror **Padding** (1D and 2D)

1. Optimizers

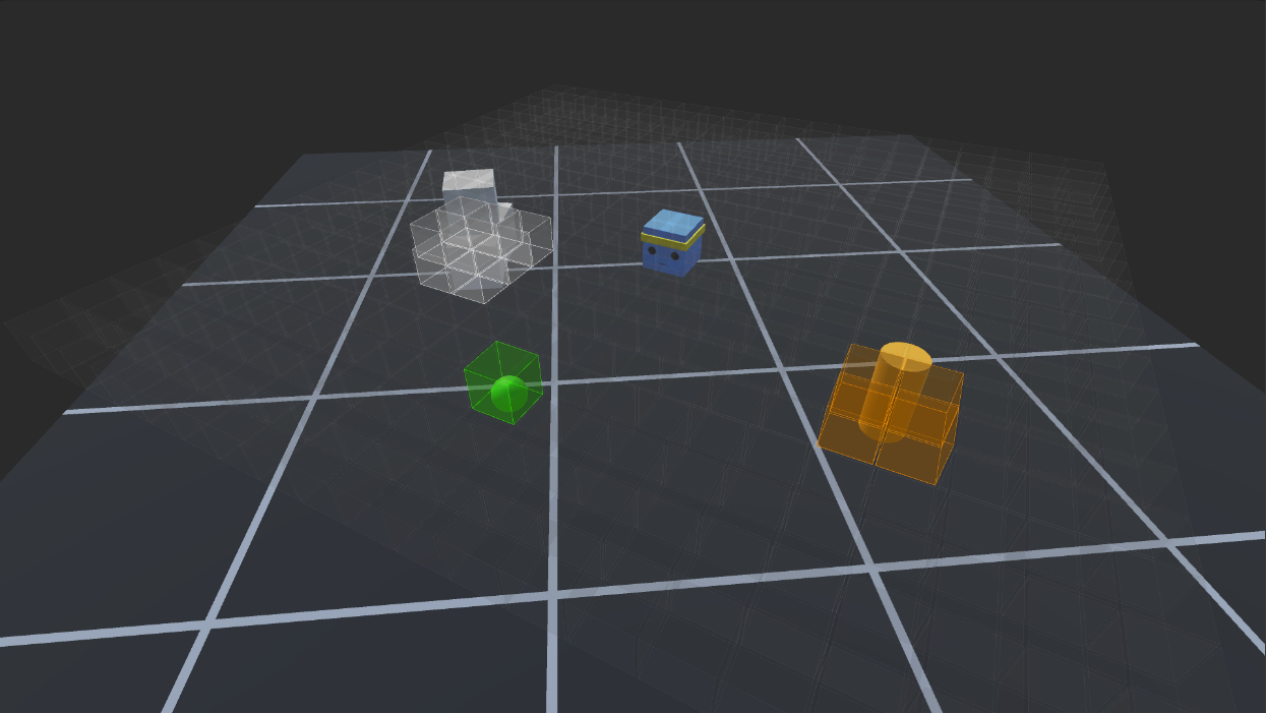
* **SGD** (+Momentum, +Nesterov)
* **RMSProp**
* **Adagrad**
* **Adam** (+**AdamW**, +**Nadam**, +**Adan**, +**Adamax**, +AMSgrad)
* **Adadelta**
* **Lion**

For reinforcement learning, the framework provides a system of sensor modules:

* **Ray sensor** - a component that casts multiple rays, retrieving information about the types and the distance to the proximal objects,

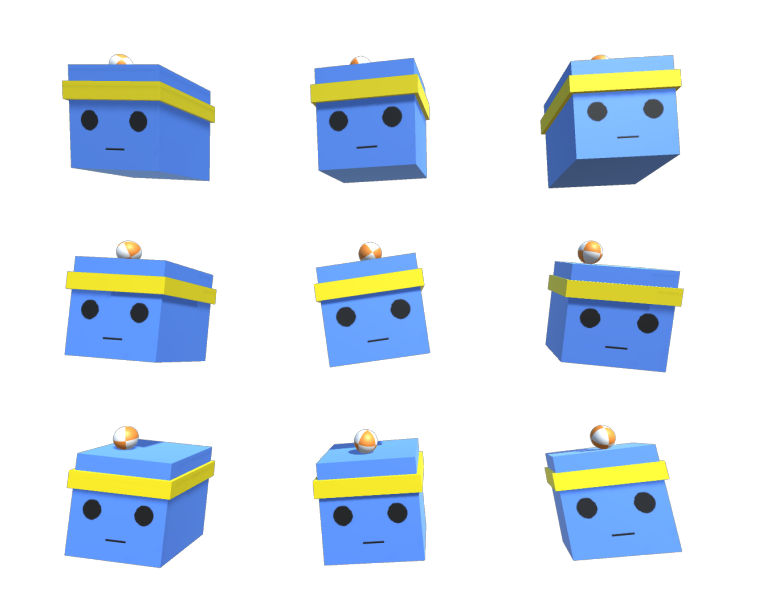
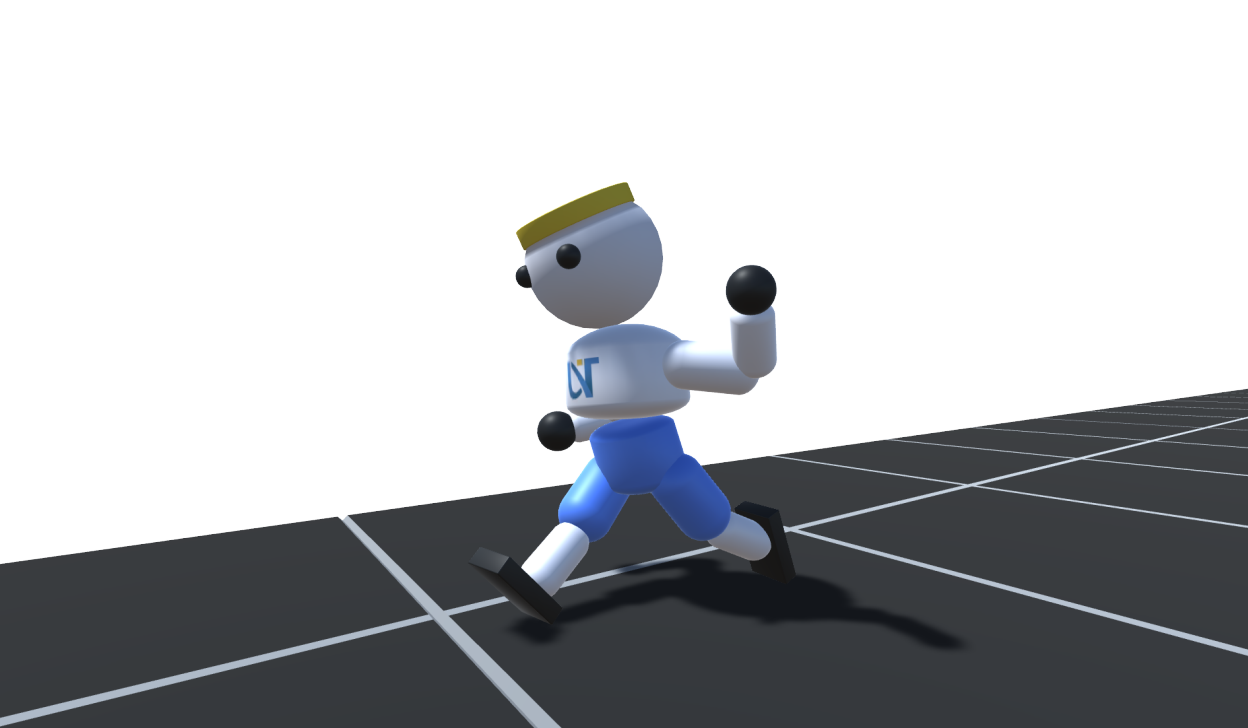
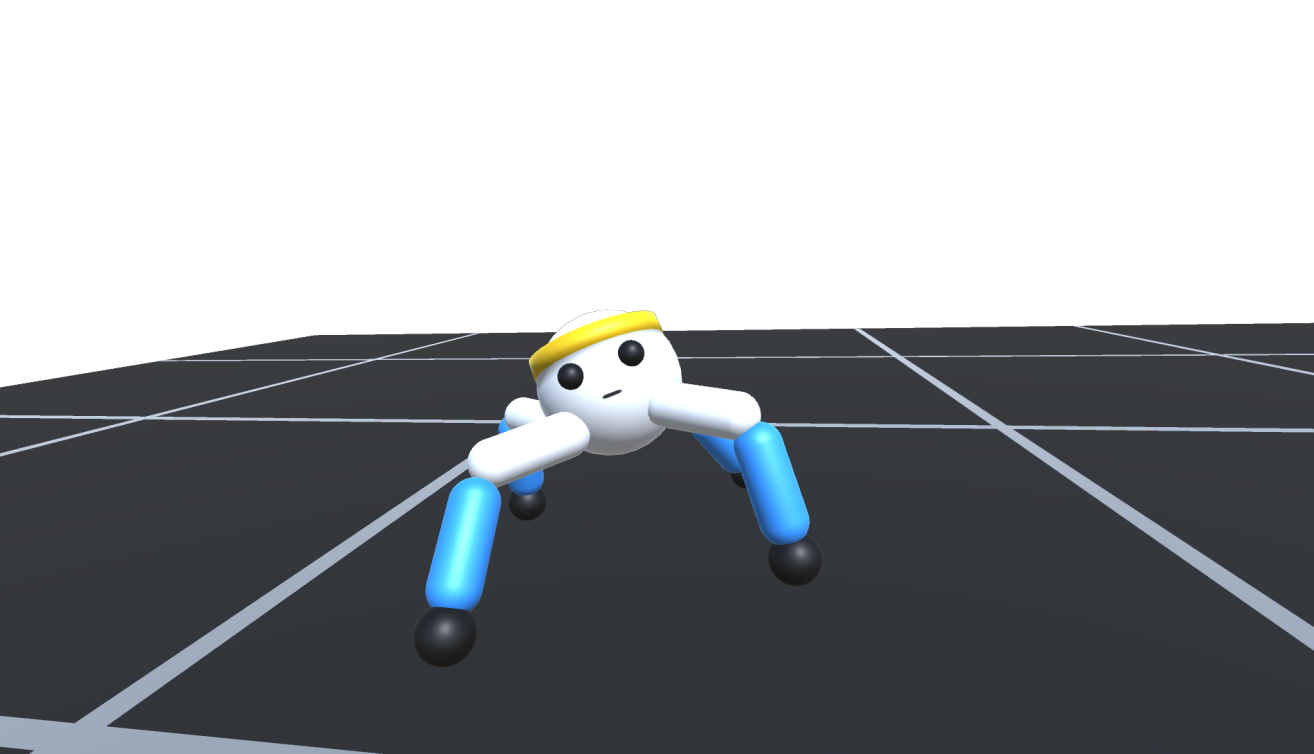


* **Grid sensor** - a component that casts a box of cells, retrieving information about the position in 3D space and type to the objects in proximity.



* **Camera sensor** - a component that uses a Unity camera to retrieve visual information for the agent (used in conv nets)

DeepUnity framework includes an extensive collection of environments[[7]](#footnote-6) responsible to highlight what it is possible to recreate and train with the current implemented features, serving as well as templates for other environments and experiments on different algorithms.



*Above is a brief showcase of some environments (from left to right: Crawler, Walker, Ball Balancer and Sorter) from [1] to showcase the results obtained with DeepUnity.*

## Conclusion

Considering the current advancements and achievements, it can be stated that Reinforcement Learning is an exploratory process of both the agent and the trainer. The current algorithms remain susceptible to the hyperparameters tuning, and a universal solution applicable to all scenarios has yet to be found. Despite that ”Reward is enough”, the requirements and computational demands do not scale proportionally with Trial and Error approaches, so, for now, it can be regarded as a complement for refining models, such as Reinforcement Learning with Human Feedback (RLHF). The AGI research entails modular enhancements and algorithms, each possessing unique strengths and weaknesses (such the example with self-attention and it’s linear complexity replacements), therefore, a robust path towards developing advanced intelligent systems involves integrating the benefits of effective architectures within modern methodologies like Joint Embeddings [7] or other that might have to come from the research community.

## References

| [1] | R.F. Ciobanu, O. Zaharie, „Policy Gradient Methods: Insights and Optimization Strategies”, *Bachelor’s Thesis, [Online],*  2024. |
| --- | --- |

| [2] | J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Kilmov, „Proximal Policy Optimization algorithms”, arXiv preprint arXiv:1707.06347, 2017. |
| --- | --- |

| [3] | T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, „Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”, In International conference on machine learning, pages 1861–1870. PMLR, 2018. |
| --- | --- |

| [4] | J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, „High-dimensional continuous control using generalized advantage estimation”, arXiv preprint arXiv:1506.02438, 2015. |
| --- | --- |

| [5] | K. O Stanley and R. Miikkulainen, „Evolving neural networks through augmenting topologies”, Evolving neural networks through augmenting topologies. Evolutionary computation, 10(2):99–127, 2002. |
| --- | --- |

| [6] | R. F. J. Dossa, S. Huang, S.Ontanon, and T. Matsubara, “An empirical investigation of early stopping optimizations in proximal policy optimization”, IEEE Access, 9:117981-117992, 2021. |
| --- | --- |

| [7] | Yann LeCun, „A path towards autonomous machine intelligence”, version 0.9. 2, 2022-06-27. Open Review, 62(1), 2022. |
| --- | --- |

1. <https://github.com/smtmRadu/DeepUnity> [↑](#footnote-ref-0)
2. <https://github.com/smtmRadu/GodotAI> [↑](#footnote-ref-1)
3. <https://github.com/smtmRadu/DeepUnity/blob/main/Assets/DeepUnity/ReinforcementLearning/Base/PPOTrainer.cs> [↑](#footnote-ref-2)
4. <https://spinningup.openai.com/en/latest/algorithms/sac.html> [↑](#footnote-ref-3)
5. <https://github.com/smtmRadu/DeepUnity/blob/main/Assets/DeepUnity/ReinforcementLearning/Base/SACTrainer.cs> [↑](#footnote-ref-4)
6. <https://github.com/Unity-Technologies/ml-agents> [↑](#footnote-ref-5)
7. <https://github.com/smtmRadu/DeepUnity/tree/main/Assets/DeepUnity/Tutorials> [↑](#footnote-ref-6)