Deep Reinforcement Learning for Rhythm Game Control

Everett Lee
Department of Computer Science
Stanford University
everettl@stanford.edu

Abstract

This project explores the use of deep reinforcement learning (RL) to train agents capable of playing the rhythm-based game *osu!catch*. Unlike traditional RL benchmarks such as Atari or MuJoCo, *osu!catch* presents a hybrid challenge: agents must combine spatial control with fine-grained temporal precision. Actions must align with musically-timed fruit spawns under tight latency constraints, making this environment ideal for studying RL performance in high-frequency, real-time domains.

We developed a custom Gym-style environment that parses `.osu` beatmap files and simulates the game at 60 frames per second. Instead of raw visual input, agents receive a four-dimensional structured observation vector: time-to-fruit, relative distance, catcher velocity, and an urgency flag. The action space includes five discrete actions—`none`, `left`, `right`, `left_dash`, and `right_dash`—which affect the catcher's motion. Rewards are shaped using a Gaussian centered on the fruit's position, modulated by urgency and penalties for unnecessary movement, jitter, and mistimed dashes. Reward shaping parameters were tuned manually for stability and responsiveness.

We trained two deep RL agents: an actor-critic model and a simplified Rainbow DQN. The actor-critic agent uses advantage-based updates and entropy regularization to balance exploration and exploitation. Rainbow incorporates dueling networks, NoisyLinear layers, and Double Q-learning to improve value estimation and policy robustness. Both agents were trained on four beatmaps of increasing difficulty and evaluated on two unseen maps (one easy, one hard). Each was trained for 1000 episodes, with hyperparameters selected independently to ensure fairness.

Rainbow DQN achieved an average reward of 2,538 on held-out maps, compared to 2,118 for actor-critic. While performance was similar on easier maps, Rainbow significantly outperformed on harder ones requiring precise timing and quick decisions. Replay visualizations confirmed that Rainbow learned more deliberate behaviors—dashing earlier, minimizing jitter, and centering under fruit—while actor-critic often hesitated or overreacted. These differences were most evident on difficult maps, where small execution errors led to larger penalties.

Our findings suggest that value-based RL methods with architectural enhancements are well-suited for rhythm-based tasks requiring high temporal resolution and coordination. This work opens opportunities to explore sequence models (e.g., Decision Transformers), extend the environment to support droplets and hyperdashes, and incorporate visual input for more human-like gameplay. More broadly, it shows how simple RL setups can generalize to structured, temporally-sensitive domains.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

Brief Abstract

We apply deep reinforcement learning to train agents to play *osu!catch*, a rhythm-based game requiring both spatial accuracy and precise timing. Using a custom Gym-style environment with structured state inputs, we compare a baseline actor-critic agent to a simplified Rainbow DQN incorporating dueling networks, NoisyLinear layers, and Double Q-learning. Both agents are trained on four beatmaps of increasing difficulty and evaluated on two held-out maps. Rainbow DQN outperforms actor-critic, particularly on harder maps, achieving an average reward of 2,538 versus 2,118. Our results highlight the suitability of value-based methods with architectural enhancements for rhythm-sensitive control tasks.

1 Introduction

Rhythm games present a unique challenge for reinforcement learning (RL): they combine real-time control, precise spatial alignment, and temporally synchronized actions. Among them, osu!catch offers a particularly interesting testbed due to its simple action space yet demanding timing and coordination requirements. In this game, players must move a character horizontally to catch falling objects ("fruits") in sync with music, requiring both rhythmic understanding and swift decision-making.

While deep RL has shown strong performance in classic control problems and visually rich games, its application to rhythm-based domains remains relatively underexplored. This project investigates whether modern RL algorithms can effectively learn to play osu!catch using structured, non-visual input. Specifically, we develop a custom Gym-style environment that models the game dynamics and train two types of agents: a lightweight actor-critic model and a more expressive Rainbow DQN.

By evaluating performance across beatmaps of varying difficulty, we aim to understand how architectural choices and input representations affect learning outcomes. Our results provide insights into which RL techniques are best suited for environments that demand fine-grained, music-aligned control.

2 Related Works

As deep reinforcement learning gained traction following the success of DQN (Mnih et al. (2013)), researchers introduced architectural refinements to address sample inefficiency, exploration, and stability. One major advancement is Rainbow DQN (Hessel et al. (2017)), which integrates several extensions—such as dueling network architectures, prioritized experience replay, double Q-learning, and noisy linear layers—to significantly enhance learning performance across Atari benchmarks. Inspired by this, we explored Rainbow DQN as a stronger alternative to our initial actor-critic baseline, particularly because of its potential for more stable and deliberate control in high-frequency decision-making settings like rhythm games.

Actor-critic methods such as A2C and PPO (Schulman et al. (2017)) are known for their lower variance and on-policy learning stability. Our original approach used an advantage-based actor-critic model with entropy regularization for smoother policies. However, we noticed that the agent's behavior often became overly reactive—an issue noted in prior work (Henderson et al. (2017)), where actor-critic methods may underperform when rewards are sparse or delayed.

Unlike most deep RL applications that rely heavily on visual input (e.g., Atari, Dota 2, StarCraft II), our environment instead uses structured, low-dimensional state vectors derived from beatmaps. This decision was motivated by prior work in state abstraction , which shows that removing visual noise can improve sample efficiency and help isolate model differences. While there is limited work specifically on RL for rhythm games, our setup mirrors environments where timing and trajectory precision are critical, such as robotic manipulation or continuous control tasks in MuJoCo.

Overall, our work builds on the success of actor-critic and Rainbow DQN methods, but adapts them to a novel and underexplored setting. By focusing on structured input and rhythm-aligned feedback, we aim to shed light on how RL methods generalize to environments requiring temporal coordination and reactive control.

3 Methods

3.1 Environment and Observations

We implemented a custom Gym-style environment that simulates *osu!catch* dynamics by parsing .osu beatmap files. At each timestep t (corresponding to 1 frame, or 16 ms ($\frac{1000 \text{ ms}}{1 \text{ second}} \cdot \frac{1 \text{ second}}{60 \text{ frames}}$), the agent receives a structured observation vector:

- dt_t: normalized time until the next fruit spawns
- dx_t: relative horizontal distance between the catcher and the upcoming fruit
- v_t: current horizontal velocity of the catcher
- u_t: binary urgency flag indicating imminent fruit spawn ($u_t = 1$ if $dt_t < \epsilon$)

The action space consists of 5 discrete actions: `none`, `left`, `right`, `left_dash`, and `right_dash`. Actions influence the position and velocity of the catcher.

The reward at time t is composed of several components, each targeting a specific behavior observed during training. These components were empirically selected and tuned based on preliminary experiments to balance catch accuracy, movement efficiency, and training stability.

- **Base reward:** Encourages alignment between the catcher and the incoming fruit. If the catcher is within a small radius $r_p = 15$, a flat reward of 1 is applied (reduced slightly if the agent is moving unnecessarily). Otherwise, a Gaussian reward is used to smoothly decay with distance:
$$R_{\text{base}} = \begin{cases} 1 - 0.02 \cdot \mathbb{I}[\Delta x_t > 0], & \text{if } |dx_t| \leq r_p \\ \exp\left(-\frac{(dx_t)^2}{2\sigma^2}\right), & \text{otherwise} \end{cases}$$
where $\Delta x_t = |x_t - x_{t-1}|$ is the magnitude of movement since the previous step.
- **Urgency scaling:** Modulates the base reward based on how soon the fruit will fall. As dt_t approaches 0, urgency increases, boosting the reward:
$$R_{\text{urgency}} = \exp\left(-\frac{dt_t}{250}\right)$$
This encourages the agent to focus more strongly on imminent fruits.
- **Directional bonus:** Rewards the agent for reducing its distance to the fruit and penalizes movement away from nearby fruits:
$$b_{\text{dir}} = \begin{cases} +0.07, & \text{if reducing distance to fruit} \\ -0.02, & \text{if increasing distance and } |dx_t| < 100 \\ 0, & \text{otherwise} \end{cases}$$
This provides a simple heuristic to promote momentum in the correct direction.
- **Movement penalty:** Penalizes excessive horizontal movement, especially when the fruit is nearby. The penalty scales with velocity and proximity:
$$P_{\text{move}} = -0.2 \cdot \frac{\Delta x_t}{128} \cdot (1 + 0.5 \cdot \text{scale}(dx_t))$$
This helps reduce jitter and prevents the agent from constantly overshooting or oscillating.
- **Jitter penalty:** Applies a small penalty when the agent changes actions too frequently, particularly when the fruit is close:
$$P_{\text{jitter}} = -0.02 \cdot \mathbb{I}[a_t \neq a_{t-1}] \cdot (1 + \text{scale}(dx_t))$$
This discourages indecisiveness and enforces smoother sequences of actions.
- **Dash penalty:** Penalizes dashing when it is not needed. The penalty increases when dashing near the fruit, where short-range adjustments would suffice:
$$P_{\text{dash}} = \begin{cases} -0.05 \cdot \text{sigmoid}(dx_t), & \text{if dashing} \\ -0.2, & \text{if dashing and } |dx_t| < 10 \\ 0, & \text{otherwise} \end{cases}$$

The final reward is the sum of these terms:

$$r_t = R_{\text{base}} \cdot R_{\text{urgency}} + b_{\text{dir}} + P_{\text{move}} + P_{\text{jitter}} + P_{\text{dash}}$$

This formulation prioritizes being correctly positioned at the right time, while penalizing erratic or excessive motion. In early experiments, these components—especially the urgency and jitter penalties—were helpful in reducing twitchy or overly reactive behavior, though the model still produces more jittery results than human players.

3.2 Actor-Critic Agent

Our baseline is an advantage-based actor-critic model, following the Advantage Actor-Critic (A2C) framework (Mnih et al. (2016)). The policy $\pi_\theta(a|s)$ and value function $V_\phi(s)$ share a common encoder network consisting of two hidden layers with 64 ReLU units each.

The actor loss encourages actions that lead to higher-than-expected returns, based on the advantage $G_t - V_\phi(s_t)$:

$$\mathcal{L}_{\text{actor}} = -\log \pi_\theta(a_t \mid s_t) \cdot (G_t - V_\phi(s_t))$$

where G_t is the total discounted return from time t onward:

$$G_t = \sum_{k=0}^{T} \gamma^k r_{t+k}$$

The critic loss minimizes the squared error between the predicted value and the observed return:

$$\mathcal{L}_{\text{critic}} = \left(G_t - V_\phi(s_t)\right)^2$$

To encourage exploration and avoid premature convergence to deterministic policies, we include an entropy regularization term:

$$\mathcal{L}_{\text{entropy}} = -\sum_a \pi_\theta(a \mid s_t) \log \pi_\theta(a \mid s_t)$$

The entropy weight was fixed at $\beta = 0.01$ throughout; while we did not conduct detailed experiments, this value appeared sufficient to maintain exploration across maps.

The total loss is then given by:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{actor}} + \mathcal{L}_{\text{critic}} - \beta \cdot \mathcal{L}_{\text{entropy}}$$

This formulation balances learning an accurate value function, reinforcing advantageous actions, and maintaining policy stochasticity for better exploration.

3.3 Rainbow DQN Agent

We implemented a simplified Rainbow DQN agent using three key improvements from the original formulation Hessel et al. (2017): dueling networks, NoisyLinear layers, and Double Q-learning. These enhancements improve exploration, learning stability, and credit assignment in value-based reinforcement learning—particularly important in rhythm-based environments like *osu!catch*, which feature sparse rewards and rapid transitions between states.

Our environment naturally lends itself to discrete action-value methods. Each timestep presents a well-defined, low-dimensional state (time to fruit, relative distance, velocity, urgency), and actions are discrete: move left, right, dash, or remain still. Q-learning is well-suited here, as the agent can associate expected returns with state-action pairs, and learn optimal movement strategies through value iteration, even in the absence of an explicit model.

- **Dueling Networks:** The Q-network is split into two streams after an initial shared feature extractor: one for estimating the state value $V(s)$ and another for estimating the action-specific advantages $A(s, a)$. These are combined using:

$$Q(s, a) = V(s) + \left(A(s, a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a') \right)$$

In our environment, this separation is especially beneficial: many frames offer little difference in value between actions (e.g., when no fruit is imminent), while other moments require highly differentiated responses (e.g., initiating a dash). The dueling architecture helps focus learning on the state's overall urgency versus the specific movement needed.

- **Noisy Layers:** We replace standard linear layers in the value and advantage streams with NoisyLinear layers (Fortunato et al. (2017)), which inject learned, parameterized noise into the weights during training. This allows the agent to perform directed exploration based on its uncertainty, rather than relying on undirected ϵ-greedy strategies. In the context of the game, this enables more efficient discovery of when to initiate dashes or hold position, especially during early training when the action landscape is uncertain.
- **Double Q-Learning:** To mitigate overestimation bias in Q-learning, we decouple the action selection and evaluation steps. The next action is selected using the online network, but its value is evaluated using the target network:

$$Q(s,a) \leftarrow r + \gamma Q_{\text{target}}\left(s', \arg\max_{a'} Q_{\text{online}}(s', a')\right)$$

  In our environment, fruits appear and disappear quickly, and timing misalignments can lead to sharply different rewards. Reducing bias in Q-value estimates ensures that rare but high-reward sequences (e.g., perfect dashes to distant fruit) are not prematurely ignored.

Taken together, these modifications enable our Rainbow DQN agent to efficiently explore, prioritize urgent state-action decisions, and maintain stability during training in the game's environment.

4 Experimental Setup

We implemented a custom Gym-style RL environment to simulate *osu!catch* using parsed `.osu` beatmap files. At each 16ms timestep, the agent receives a 4D observation vector: normalized time-to-fruit (dt), relative horizontal distance (dx), catcher velocity, and a binary urgency flag. The action space consists of five discrete options: `none`, `left`, `right`, `left_dash`, and `right_dash`, corresponding to movement and dash decisions commonly made by human players.

Rewards follow a Gaussian centered at the fruit's x-position and are scaled by urgency to emphasize timing-critical moments. We also apply additional penalties for excessive movement, jitter, and mistimed dashes, which we found to be important for reducing erratic behaviors. A reward plateau within 15 pixels of the fruit promotes positional stability once close enough to the fruit.

Both agents are trained on a fixed set of four beatmaps spanning beginner to expert difficulty levels and evaluated on two held-out maps (one easy, one hard). Each model is trained for 1,000 episodes. The actor-critic agent completes training in approximately 40 minutes, while Rainbow DQN—due to the overhead of replay sampling and target network computation—requires approximately 8 hours. Performance is evaluated based on average episode reward on the two test maps.

Training strategies differ by agent type. The actor-critic model is trained using full-episode rollouts with return-based updates and advantage-weighted policy gradients. In contrast, Rainbow DQN samples uniformly from a replay buffer of 10,000 transitions and performs minibatch temporal difference updates. A target network is synchronized every 100 environment steps to stabilize learning.

Model-specific hyperparameters (e.g., learning rate, entropy coefficient, noise standard deviation) were selected independently for each agent based on early pilot runs and fixed throughout training to maintain stability and enable fair comparison.

As a summary, we compare the following two agent architectures:

- **Actor-Critic:** An advantage-based A2C agent with a shared MLP encoder consisting of a 64-unit ReLU layer, followed by separate policy and value heads. The model is trained end-to-end using policy gradient loss with entropy regularization.
- **Rainbow DQN:** A value-based agent that incorporates dueling network heads, NoisyLinear layers for learned exploration, and Double Q-learning updates. The network samples transitions in minibatches and applies mean-squared error loss on the temporal difference target.

5 Results

We trained both agents for 1,000 episodes on a set of four beatmaps spanning beginner to expert difficulty. Evaluation was conducted on two previously unseen maps—one medium and one hard—to assess generalization performance. Average test rewards were computed by averaging the final smoothed reward across multiple episodes per map. Rainbow DQN achieved an average reward of 2,538, while the actor-critic model reached 2,118, indicating a clear performance advantage for the value-based agent.

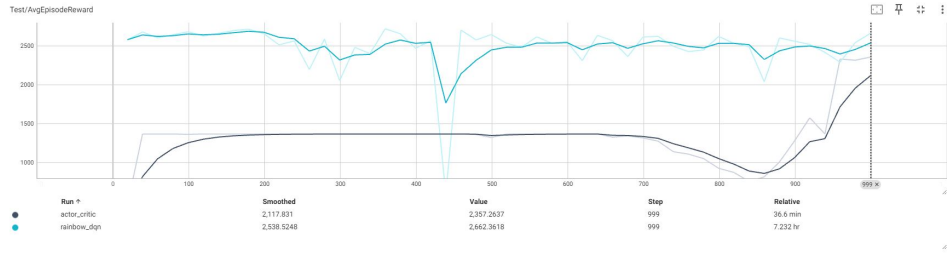

Figure 1: Evaluation episode rewards on held-out test medium and hard beatmaps. Rainbow DQN maintains higher performance and more consistent generalization.

Rainbow DQN exhibited strong learning dynamics throughout training. As shown in Figure 2, it began outperforming actor-critic early in training and maintained a consistent advantage over it even as actor-critic improved. This may be due to the use of NoisyLinear layers and experience replay, allowing the Rainbow DQN model to explore and retain beneficial behaviors efficiently. In contrast, the actor-critic model displayed noisier training curves with slower reward accumulation and frequent regressions, especially during transitions to harder maps.

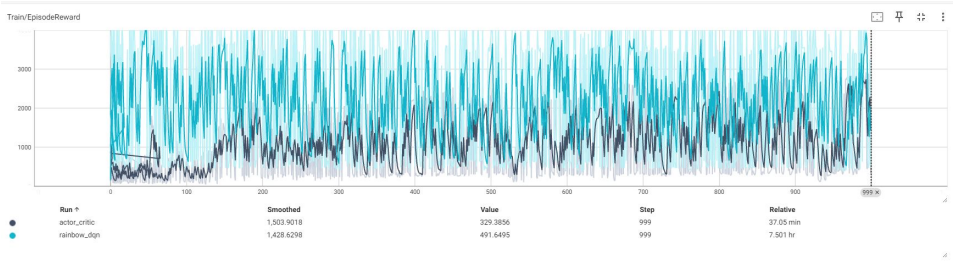

Figure 2: Training rewards over 1,000 episodes. While results are relatively unstable throughout, rainbow DQN generally outperforms actor-critic at most timesteps.

Although the Rainbow DQN model required significantly longer wall-clock training time, its performance advantage persisted even when controlling for compute budget. At the 36-minute mark, Rainbow had already surpassed the actor-critic model in reward accumulation, suggesting that its architectural improvements lead to greater sample efficiency and faster policy refinement despite slower per-episode throughput.

Qualitative analysis of replay visualizations supports these trends. While the Rainbow DQN agent still exhibited noticeable jitter, its movements were comparatively smoother and less erratic than those of the actor-critic agent. Rainbow tended to dash unnecessarily less often and demonstrated slightly better temporal alignment when responding to fast or clustered fruit patterns. Although positional jitter remained, it was less abrupt and more visually coherent, suggesting modest improvements in motion stability and anticipation.

In contrast, the actor-critic agent frequently overreacted to individual fruit positions, producing sharp, twitchy direction changes and delayed dashes. This often led to fruit misses, particularly in difficult sections where rapid and precise reactions were necessary.

Notably, Rainbow DQN also appeared more resilient when misaligned—it recovered from positioning errors more gracefully, adjusting its trajectory in time to catch subsequent fruits. The actor-critic

model, by comparison, struggled to recover from early missteps and often failed to realign quickly enough in high-density sequences.

While both agents demonstrated some generalization to unseen maps, Rainbow's architectural enhancements—such as NoisyLinear exploration and stable value estimation—enabled more consistent learning and robustness under time-critical conditions (e.g., low dt). These results highlight the value of augmented value-based methods in rhythm-based control domains where temporal precision and recovery matter.

Short demonstration videos for both agents on the same beatmap can be found here:
Actor-Critic: `https://drive.google.com/file/d/19noxNscVGqRrl1v9TPxF4ZtEe1-d_YyE/view?usp=sharing`
Rainbow DQN: `https://drive.google.com/file/d/1AIN-CN7GFyr3GY0lr5ERVKTi2Ln56b0U/view?usp=sharing`

6 Discussion

While our agents demonstrated meaningful performance gains, several limitations remain. One major issue is residual jitter—frequent, low-magnitude back-and-forth motion—that remains noticeable during evaluation. Although Rainbow DQN reduced the severity and frequency compared to the actor-critic agent, it did not fully eliminate it. Future work could explore temporal smoothness penalties or recurrent policy architectures to better regulate motion continuity.

Implementing the environment itself posed substantial engineering challenges. Correctly handling juice streams, curve-based slider paths, and repeat logic required a combination of geometric interpolation and detailed timing analysis. These mechanics are complex and largely undocumented, particularly in how they interact with game physics and scoring behavior. As a result, we had to implement our best guesses of how they actually act, which adds an element of uncertainty to the accuracy of the environment.

Reward shaping and debugging presented additional obstacles. Our agents were sensitive to hyperparameter choices such as movement penalties and urgency scaling, with small changes often leading to major behavioral shifts. Since actor-critic training typically took 30–60 minutes per run, and debugging often relied on visual replay inspection, iterating on reward functions was slow. Identifying root causes of failure frequently required retraining the model, generating a replay, and manually analyzing the results in-game.

Another difficulty involved disentangling model limitations from parser inaccuracies. In some visualizations, agents appeared to misread slider paths or miss aligned fruit despite seemingly correct actions. It was unclear whether these were due to inadequate policy generalization or errors in how the environment simulated the underlying mechanics.

Finally, while both models achieved competent performance, they still fall short of expert human players. Human gameplay is characterized by anticipatory motion, rhythm awareness, and fluid, high-precision control that current RL policies struggle to replicate. Bridging this gap may require multimodal inputs (e.g., audio-visual cues), imitation learning from top players, or temporally-aware models such as Transformers.

Overall, these challenges highlight promising directions for future work, both in refining the environment and advancing agent architectures for rhythm-based real-time control.

7 Conclusion

This work explores deep reinforcement learning for rhythm-based control using *osu!catch*, a domain requiring precise spatial-temporal coordination. We implemented a custom environment capable of simulating core gameplay elements and trained agents using both actor-critic and Rainbow DQN algorithms.

Our results show that value-based methods with architectural enhancements (e.g., dueling networks, NoisyLinear layers) outperform policy gradient methods in this high-frequency setting. Rainbow DQN achieved higher test rewards and exhibited more deliberate behavior, particularly on difficult maps.

These findings validate the potential of structured RL in rhythm games and highlight the effectiveness of architectural improvements for temporal alignment tasks. Future directions include modeling longer-term dependencies via sequence models (e.g., Decision Transformers), incorporating raw visual or auditory input to better reflect the visual and auditory cues humans rely on during gameplay, and refining reward shaping to reduce jitter and improve movement smoothness.

8 Team Contributions

This project was completed individually by me. Taran Kota (tkota@stanford.edu) contributed to the original project proposal as a group member, but did not contribute to anything else before dropping the class.

References

Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Rémi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. 2017. Noisy Networks for Exploration. *CoRR* abs/1706.10295 (2017). arXiv:1706.10295 `http://arxiv.org/abs/1706.10295`

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2017. Deep Reinforcement Learning that Matters. *CoRR* abs/1709.06560 (2017). arXiv:1709.06560 `http://arxiv.org/abs/1709.06560`

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. 2017. Rainbow: Combining Improvements in Deep Reinforcement Learning. arXiv:1710.02298 [cs.AI] `https://arxiv.org/abs/1710.02298`

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. *CoRR* abs/1602.01783 (2016). arXiv:1602.01783 `http://arxiv.org/abs/1602.01783`

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. (2013). `http://arxiv.org/abs/1312.5602` cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347 [cs.LG] `https://arxiv.org/abs/1707.06347`