RL project: Snake

Giacomo Vianello [1]

Abstract

This work applies Reinforcement Learning (RL) to the classic *Snake* game, moving from an interpretable heuristic baseline to advanced deep models. The heuristic policy provides a stable but limited baseline due to its rigid rules. To surpass these limits, we implement and compare three RL algorithms: Advantage Actor-Critic (A2C), Double Deep Q-Network (DDQN), and Proximal Policy Optimization (PPO). PPO clearly outperforms the others, delivering more stable learning and higher average scores via entropy-driven exploration and clipped objectives.

1. Introduction

This work builds on the *Snake* assignment template, refactored from TensorFlow to **PyTorch** to improve flexibility, portability, and computational efficiency. PyTorch enables a more intuitive workflow, consistent cross-platform deployment, and seamless *GPU acceleration*. On this framework, we implemented a robust RL pipeline with high-throughput parallel rollouts, offering a unified environment for comparing multiple RL architectures.

In the following subsections we briefly describe our environment, objectives, setup, and metrics.

1.1. Environment and Rewards

The environment we are provided generates a snake-game board of size $(N \times N)$, where the outer cells defines the walls and the remaining $(N-1)^2$ cells define the playable area. The agent can choose among the *discrete* set of actions

$$\mathcal{A}(s) = \{\text{UP, RIGHT, DOWN, LEFT, NONE }\},\ s \in \mathcal{S},$$

where $S \in \{0, \dots, 4\}^{N \times N}$ is the discrete set of states.

While the environment operates on a discrete grid, the task is effectively a **continuing RL problem**, as episodes only reach a natural termination when the board is entirely filled and no further fruits can be spawned. This characteristic significantly increases the complexity of the RL setup, making a definitive "win" condition exceptionally difficult to achieve under standard constraints. While a full transition to classical Snake rules would require a different environment definition, we address this challenge by shaping the reward signals to incentivize survival and growth. Consequently, our primary learning objective is the **maximization of fruit consumption within a fixed time-horizon**, while simultaneously minimizing failures such as wall collisions.

The reward function $R(s, a, s')$ is designed to prioritize long-term survival and goal-oriented behavior. The specific values used in this implementation are as follows

$$R(s,a,s') = \begin{cases} +10.0 & \text{win the game (rare)} \\ +0.5 & \text{fruit acquired} \\ -0.5 & \text{self-eating} \\ -0.1 & \text{wall collision} \\ -0.01 & \text{step penalty} \end{cases} \tag{1}$$

Our modifications transform the sparse terminal signal into a dense reward landscape that better guides the agent. By introducing a -0.01 step penalty, we discourage aimless wandering and compel the agent to seek efficient paths. Increasing the self-eating penalty to -0.5 (from -0.2) explicitly penalizes reckless maneuvers, forcing the policy to prioritize its structural integrity as the snake grows. Finally, we significantly increased the win reward $(+10.0)$ to incentivize board completion, ensuring the agent differentiates between mere survival and the ultimate goal of filling the grid. These adjustments align the optimal strategy with persistent growth and meticulous spatial management.

1.2. Action Masking

To improve training efficiency, we implement a safety mask that discourages moves leading to immediate wall collisions. This mechanism functions as an inductive bias by applying a penalty directly to the policy logits before action sampling. Mathematically, the masked logits $\tilde{l}$ are computed as:

$$\tilde{l} = l + (\alpha \cdot M)$$

where l represents the original output logits, $M \in \{0,1\}^4$ is a binary vector indicating wall-collision moves, and $\alpha = -1$

[1]Department of Information Engineering, Università degli Studi di Padova. Full code: GitHub.. Correspondence to: Giacomo Vianello <giacomo.vianello.2@studenti.unipd.it>.

Reinforcement Leaning Course, AY 2025/26.

is the penalty coefficient. This adjustment lowers the probability of sampling trivially detrimental actions, allowing the model to focus its learning capacity on complex objectives – such as fruit acquisition and self-collision avoidance – rather than the static boundaries of the environment. By pruning the action space in this manner, the learning process becomes significantly more stable and sample-efficient.

1.3. Setup and Evaluation Metrics

All agents were trained in a vectorized environment with $N_B = 1000$ parallel 7×7 boards for 20×10^6 total steps. Despite the high computational cost, GPU acceleration and efficient parallel rollouts kept training time for each model under 10 minutes. We set the discount factor to $\gamma = 0.995$ for all agents to encourage consideration of delayed rewards from distant fruit and long-term path planning, which is essential given the sparse positive rewards in the Snake environment.

Performance is benchmarked using three primary metrics: (i) **Average Reward**, representing overall policy efficiency; (ii) **Mean Fruit Consumption**, serving as a proxy for the game score; and (iii) the **Wall Collision Rate**, which acts as a critical safety metric for boundary avoidance.

2. Baseline Policy

We designed a heuristic baseline to expose the environment's core challenges and motivate using more complex Reinforcement Learning architectures.

This policy operates on a greedy principle, prioritizing the move that reduces the *Manhattan distance* between the snake's head and the fruit. However, to ensure basic survival, the policy incorporates a conditional safety check: before executing a move, it verifies if the target cell contains a wall or a segment of the snake's own body. If the greedy action is deemed unsafe, the agent samples a move from the set of available safe directions; in the event that all adjacent cells lead to a collision, the agent selects an action at random as a last-resort measure.

Evaluating the baseline policy on a 7×7 grid across varying time horizons yields the performance metrics summarized in Table 1. The heuristic demonstrates robust initial performance, maintaining a high mean fruit consumption and a relatively stable reward structure even over $10,000$ steps. These results indicate that the "greedy-plus-safety" logic is a highly effective reasonable policy for standard navigation and immediate obstacle avoidance.

The slight drop in mean reward and fruit consumption, along with the higher wall-hit rate (10.39% to 13.14%) as trial length increases, shows this approach is not exhaustive. Effective in the short term, the policy lacks the strategic depth to handle complex "self-trapping" scenarios as the snake grows. This performance plateau indicates that, although the baseline is strong, advanced Reinforcement Learning is needed for long-term survival and optimal grid use.

Table 1. Baseline performance: Mean metrics across evaluation horizons of 100, 1000, and 10000 steps in the fully observable environment.

PLAYED STEPS	REWARD	FRUITS	WALL-HITS
100	0.102814	227.79	10.39
1000	0.096466	220.48	12.74
10000	0.095873	220.24	13.14

3. RL Approaches

The heuristic baseline shows basic task competence but plateaus due to limited planning and spatial adaptability. To overcome this, we apply Deep Reinforcement Learning (DRL) to the *Snake* problem, implementing and comparing three algorithms – A2C, DDQN, and PPO – to assess convergence stability, sample efficiency, and long-term survival.

3.1. Architectural Backbone

To ensure architectural consistency and a fair comparison between algorithms, all agents share a unified Convolutional Neural Network (CNN) backbone. The state is represented as a $7 \times 7 \times 4$ one-hot encoded grid, which is permuted from a channel-last (`NHWC`) to a channel-first (`NCHW`) format to meet PyTorch's computational requirements. The architecture consists of a *feature extraction stage* with 3 convolutional layers (3×3 kernels, unit padding) utilizing 32, 64, and 128 filters with ReLU activations, followed by a 128-unit fully connected layer for *non-linear refinement*. This latent representation is then mapped to specialized heads: a single Q-value output for DDQN, and dual policy and value heads for A2C and PPO.

This configuration was specifically chosen not only for the fully observable task but to provide the representational capacity required for *partially observable environments*. In such scenarios, the increased depth and the intermediate dense layer enable the agent to better generalize from limited local views, significantly improving performance compared to shallower architectures.

3.2. Double Deep Q-Network (DDQN)

To mitigate the value overestimation bias in standard Q-learning, we use the Double DQN algorithm (van Hasselt et al., 2015), which decouples action selection from target value estimation using two identical networks: an *Online Network* (θ) and a *Target Network* (θ').

The agent operates by minimizing the Mean Squared Error

(MSE) between the predicted Q-values and the target values. In our implementation, the target Y_t is computed as

$$Y_t = r_{t+1} + \gamma Q(s_{t+1}, \underset{a}{\text{argmax}} Q(s_{t+1}, a; \theta); \theta').$$

Here, the *Online* network selects the optimal action for the next state, while the *Target* network evaluates its value. This separation significantly improves stability in the stochastic environment of the Snake game. The target network weights are updated via a "hard update" mechanism, synchronizing with the online network every $5,000$ steps.

To break the temporal correlation of sequential game data, we utilize a *Replay Buffer* with a capacity of $100,000$ transitions. During training, uniform random batches of 256 transitions are sampled to update the network.

Exploration is managed via an ϵ-greedy strategy with exponential decay. The exploration rate ϵ_t at step t is

$$\epsilon_t = \epsilon_{end} + (\epsilon_{start} - \epsilon_{end}) \cdot \exp\left(-\frac{t}{\tau}\right),$$

where $\epsilon_{start} = 0.5$ is the initial exploration rate, $\epsilon_{end} = 0.01$ is the minimum rate, and τ is the decay time constant, derived from the total training iterations to ensure a smooth transition across the learning horizon. During the action selection phase, the agent samples a random action with probability ϵ_t. Otherwise, it selects the greedy action by maximizing the Q-values, which are augmented by the safety mask penalty to prune illegal moves

$$a_t = \underset{a}{\text{argmax}} \left(Q(s_t, a; \theta) + \alpha \cdot M\right).$$

Here, M is the binary safety mask and $\alpha = -1$ is the penalty coefficient that discourages immediate wall collisions.

3.2.1. DDQN Training Results

The DDQN agent was trained for 20 million environmental steps; however, for logging efficiency, metrics were aggregated across the $N_B = 1000$ parallel boards at each iteration. Consequently, the training curves in Figure 1 are plotted against 20,000 aggregated training steps, where each unit represents a full vectorized update.

The results shown in Figure 1 underscore the inherent instability of value-based methods in this domain. While the mean reward quickly surpasses the heuristic baseline, it exhibits high variance throughout the training horizon. This instability is reflected in the wall collision metric, which fails to converge to zero even in the final stages of training. This residual error is likely a byproduct of the ϵ-greedy strategy; despite exponential decay, the stochastic nature of action selection ensures that the agent occasionally executes high-risk moves.

In contrast, the Fruit Consumption shows a consistent upward trend, stabilizing at approximately 270 units – well above the baseline. This confirms the model's ability to learn spatial navigation for goal pursuit. The plateau in the final third of training indicates that ϵ-greedy exploration and hard target updates hit a performance ceiling, failing to fully capture the strategic complexity of the 7×7 grid.

3.3. Advantage Actor-Critic (A2C)

The A2C algorithm is an on-policy method that combines value-based and policy-gradient approaches. Unlike DDQN, which estimates action values, A2C uses two heads: an *actor* that outputs an action distribution $\pi(a|s)$ and a *critic* that estimates the state-value function $V(s)$.

Our implementation leverages the synchronous version of the algorithm, utilizing N_B parallel environment instances to collect and process transitions simultaneously. The agent optimizes a composite loss function

$$\mathcal{L} = \mathcal{L}_{\text{policy}} + c_1 \mathcal{L}_{\text{value}} - c_2 \mathcal{H},$$

where $c_1 = 0.5$ and $c_2 = 0.05$ are coefficients for the value loss and entropy regularization, respectively. The policy loss is defined as $\mathcal{L}_{\text{policy}} = -\mathbb{E}[\log \pi(a|s)\delta]$, where the TD error $\delta = r + \gamma V(s') - V(s)$ serves as a proxy for the advantage. By scaling the log-probabilities with δ, the agent increases the likelihood of actions that outperform the state average. The value loss is computed as $\mathcal{L}_{\text{value}} = \mathbb{E}[\delta^2]$, which minimizes the mean squared error between the critic's state-value estimate and the bootstrapped return. Finally, the entropy term $\mathcal{H}$ prevents premature convergence by penalizing overly deterministic distributions, thereby sustaining exploration.

As with the other agents, the A2C policy logits are augmented with a safety mask penalty ($\alpha = -1$) to prune wall-collision trajectories during action sampling.

3.3.1. A2C Training Results

Training results for the A2C agent (Figure 2) show much faster convergence on primary objectives than DDQN (Figure 1). Within 4,000 steps, A2C prioritizes foraging: fruit consumption stabilizes near 275 units per iteration—about 20% above the greedy baseline—while wall collisions quickly drop to near zero and remain there. The mean reward rises steadily and soon exceeds the heuristic but shows sharp drops in reward and spikes in collisions, likely due to the high variance of synchronous actor-critic updates in complex self-trapping states.

By contrast, the DDQN agent (Figure 1) learns more smoothly but much more slowly. It eventually surpasses the baseline in reward and fruit consumption but never fully eliminates wall collisions, indicating that value-based meth-

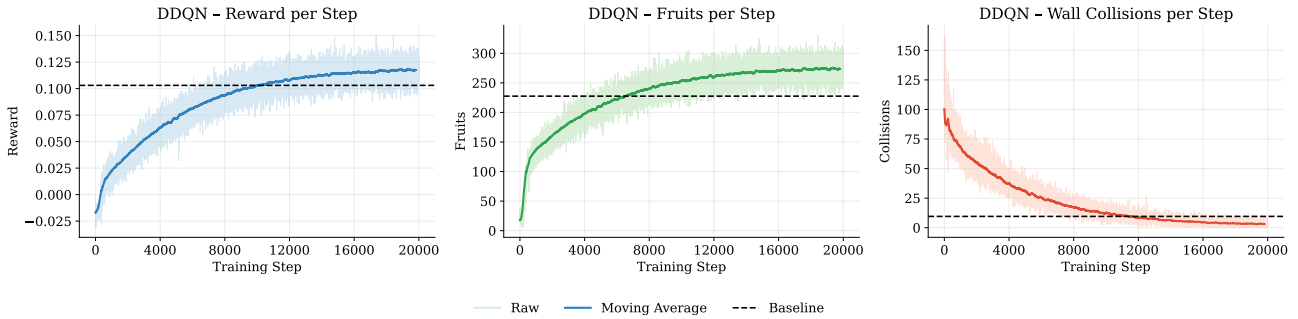

Figure 1. DDQN Training Metrics: Evaluation of Mean Reward, Fruit Consumption, and Wall Collisions. The shaded curves represent raw data, the thicker lines denote the moving average, and the dashed black lines indicate the performance of the heuristic baseline.

ods struggle more than policy-gradient approaches to encode strict safety constraints in this environment.

3.4. Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) (Schulman et al., 2017) is employed as our primary on-policy actor-critic method due to its reputation for balancing ease of implementation with robust performance. Unlike A2C, which makes one gradient update per rollout, PPO uses a clipped surrogate objective that supports multiple SGD epochs on the same batch without causing large, destabilizing policy changes.

The core of our PPO agent is the objective function that prevents the new policy π_θ from deviating too far from the old policy $\pi_{\theta_{old}}$. The probability ratio $r_t(\theta)$ is defined as

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

The agent maximizes the following clipped objective

$$\mathcal{L}^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right]$$

We set the clipping parameter to $\epsilon = 0.2$, keeping updates conservative – crucial in this continuing environment. Because the task rarely terminates naturally, the agent must maintain a highly stable policy to avoid high-penalty events (wall or self-collisions) that would otherwise cut short long-term reward accumulation within the fixed time horizon.

We compute advantages $\hat{A}_t$ using Generalized Advantage Estimation (GAE) (Schulman et al., 2015) with discount factor $\gamma = 0.995$ and GAE parameter $\lambda = 0.95$ over a 256-step rollout horizon. This lets the agent credit current actions for their impact on future rewards, effectively handling delayed gratification when navigating toward a fruit.

PPO's efficiency relies on systematic data collection and optimization. The agent uses a *Rollout Buffer* to store observations, actions, log-probabilities, rewards, and state-values from N_B parallel environments. This vectorized setup accelerates data throughput and increases sample diversity per update, stabilizing gradient estimates. We apply a linear entropy decay, reducing the entropy coefficient β from 0.05 to 0 over training to encourage broad early exploration and a progressively more deterministic, near-optimal policy. After each rollout, we optimize with Adam (learning rate 4×10^{-4}) for 4 epochs per batch, extracting more information from each trajectory while the clipped surrogate objective prevents destabilizing policy updates.

3.4.1. PPO Training Results

Training results for the PPO agent in Figure 3 show it is the most effective and stable architecture in this benchmark. Its learning curve is steep, surpassing the heuristic reward baseline within 2,000 steps and plateauing at the highest observed Mean Reward ($\simeq 0.126$). This performance stems from large, diverse batches—256 experience steps across N_B parallel environments per rollout—which produce smoother, more reliable convergence than the higher-variance updates of DDQN.

Fruit consumption stabilizes at about 285 units per iteration, the highest of all agents, while wall collisions drop to near zero and stay there. This shows that the clipped surrogate objective and safety mask quickly and permanently encode environmental constraints. By maximizing foraging efficiency with an almost perfect safety record, PPO is the best choice for the Snake task, combining fast learning with exceptional policy stability.

Despite its efficiency, the PPO training curve plateaus before achieving a "win" (filling the 7×7 grid). The likely cause is the feed-forward CNN's lack of spatial memory, which prevents the agent from tracking its growing tail in dense configurations. Without a recurrent mechanism like an LSTM to remember body segments beyond its immediate view, the agent adopts a conservative local survival strategy instead of the global path optimization needed for the endgame. Thus, PPO fully exploits a reactive policy but is ultimately constrained by the current architecture's representational limits.

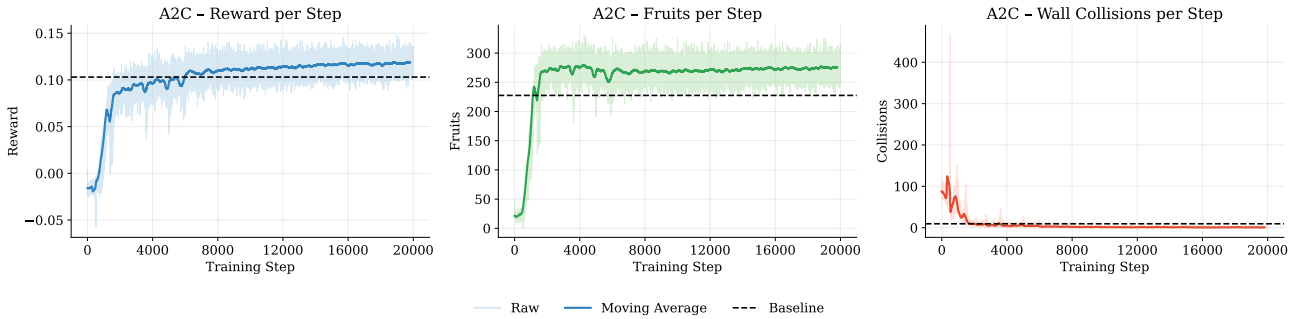

Figure 2. A2C Training Metrics: Mean Reward, Fruit Consumption, and Wall Collisions. The actor-critic approach demonstrates superior stability and a significantly lower collision rate compared to DDQN.

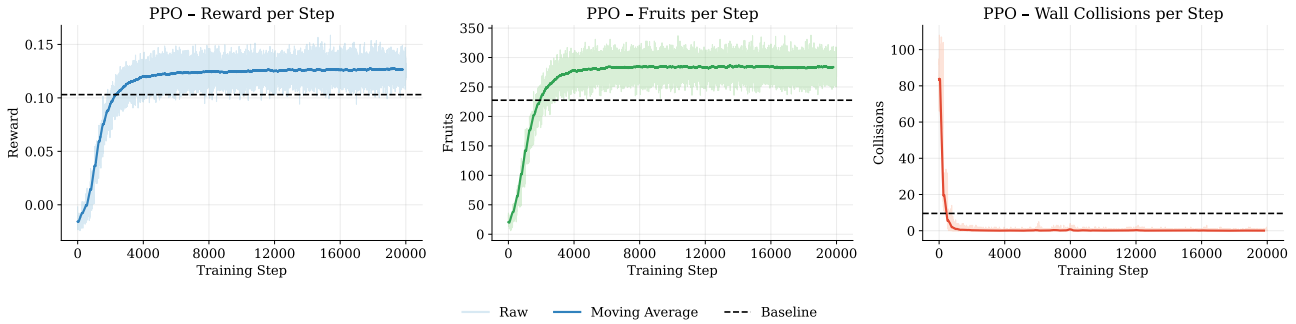

Figure 3. PPO Training Metrics: The agent achieves superior stability and reward maximization. The near-zero wall collision rate highlights the effectiveness of the clipped objective in learning safe navigation boundaries.

4. Results and comparison

To conclude the evaluation, we analyze the comparative performance of the three reinforcement learning agents against the heuristic baseline. The final metrics, derived from a 500-step evaluation in the fully observable 7×7 environment, are summarized in Table 2.

The empirical data highlights a clear performance hierarchy. PPO emerges as the superior architecture, achieving the highest mean reward of 0.1271 and a fruit consumption rate of 282.89. Most notably, PPO virtually eliminated wall collisions (0.05), indicating that its clipped surrogate objective, combined with the safety mask, successfully learns a near-perfect boundary-avoidance policy. A2C also demonstrates robust performance, maintaining a high reward (0.1185) and reducing collisions to just 2.6, significantly outperforming the baseline.

In contrast, the results for DDQN reveal a critical instability. While it successfully learns to forage—surpassing the baseline in fruit consumption (243.98 vs. 221.36)—its failure to internalize safety constraints is catastrophic. With a massive wall-hit rate of 141.86, the penalties accumulated from collisions drag its overall mean reward (0.0964) below that of the heuristic baseline (0.0971). This confirms that in this continuing environment with soft constraints, the value-based agent struggles to balance aggressive food-seeking with long-term survival, resulting in a policy that is technically "smarter" at finding paths but functionally more reckless than a simple greedy heuristic.

Table 2. Evaluation results: Mean metrics for baseline and RL agents in the fully observable environment over 500 steps.

AGENT	MEAN REWARD	MEAN FRUITS	WALL-HIT
BASELINE	0.09711	221.36	12.41
DDQN	0.09635	243.98	141.86
A2C	0.11846	274.47	2.6
PPO	**0.12708**	**282.89**	**0.05**

As shown in Figure 4, the training trajectories further differentiate the paradigms. PPO achieves the most stable learning curve, characterized by monotonic reward growth and rapid convergence to near-zero collisions. A2C initially matches PPO's foraging efficiency but suffers from periodic instability—visible as sharp reward drops and collision spikes—likely due to the high variance of synchronous updates. Finally, DDQN exhibits the slowest convergence profile and fails to fully eliminate wall collisions even by the end of the training horizon. These results validate PPO as the optimal architecture for the Snake environment, as it consistently balances high-throughput foraging with the reliable policy stability required for long-term survival.

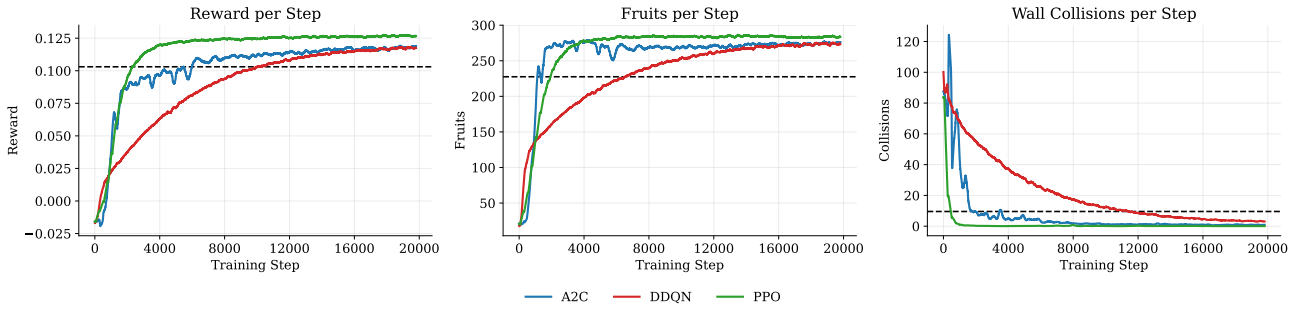

Figure 4. Cross-Algorithmic Training Comparison: Comparative training trajectories for A2C, DDQN, and PPO.

5. Partial Observability

To assess PPO's robustness under information asymmetry, we compared its performance in a Fully Observable environment to a Partially Observable one. In the partial setting, the agent sees only an egocentric 3×3 window around the snake's head. As shown in Table 3, we use the heuristic Baseline as a control; because it reads the full board state, its performance is constant and serves as a "perfect information" upper bound.

The PPO (par.) model was retrained from scratch on the partial view, since switching from an allocentric (fixed-grid) to an egocentric (snake-centered) representation requires learning new spatial semantics. Directly applying the original model to the partial view caused immediate policy collapse, whereas retraining enabled adaptation to the restricted input.

The results show a clear split in spatial reasoning. Local Survival is mostly unchanged: PPO (par.) maintains a wall-hit rate of 0.200, indicating the CNN backbone learns obstacle avoidance as a local control task. However, there is a substantial Cost of Blindness in foraging efficiency: mean fruit consumption falls from 282.45 to 174.2 (about 38%). Without a global view of the 7×7 grid, the agent cannot plan optimal paths to distant rewards and instead defaults to conservative local search. Thus, local observability suffices for survival, but global state information is crucial for efficient path planning.

Table 3. Comparative performance metrics: Mean results for Baseline and PPO agents across Full and Partial (3×3 mask) observability over 500 evaluation steps.

AGENT	REWARD	MEAN FRUITS	WALL-HIT
BASELINE (FULL)	0.097	220.67	12.64
PPO (FULL)	0.127	282.45	0.07
BASELINE (PAR.)	0.097	221.5	12.23
PPO (PAR.)	0.081	174.2	0.200

6. Conclusions and Improvements

This project implemented and benchmarked deep reinforcement learning agents in a high-throughput, vectorized Snake environment, identifying PPO as the most effective architecture. While the baseline heuristic offered a stable foundation, it lacked the adaptive reasoning required for long-term survival in dense configurations.

The architecture's main strength is computational efficiency: using PyTorch and parallel environments, we simulated millions of steps in under ten minutes with GPU acceleration. Action masking was crucial for policy-gradient agents, letting them skip learning physical boundaries and focus on survival strategies. As a result, PPO and A2C consistently achieved high fruit consumption and near-zero wall hits. In contrast, value-based DDQN failed to learn these safety constraints, showing a reckless, high-variance policy that erased its foraging gains. This indicates that policy-gradient methods are better suited for environments requiring strict safety constraints alongside reward maximization.

A general limitation remains: the game is rarely "beaten" (filling the 7×7 grid). The continuing MDP structure, while stable for training, yields conservative policies that lack the aggressive spatial optimization required to solve the endgame. Additionally, the transition to partial observability revealed a sharp "Cost of Blindness," where the feed-forward CNN lacked the spatial memory to navigate effectively without a global view.

To enable the agent to fully fill the grid, the most critical refinement would be transitioning to a terminal MDP, where collisions immediately end the episode. This would align the agent's incentives with the high-stakes survival required for board completion. Architecturally, replacing the standard CNN with LSTMs would mitigate the partially observable trade-off, while implementing Asynchronous Advantage Actor-Critic (A3C) could further stabilize gradient updates through asynchronous parallelization.

References

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015. URL http://arxiv.org/abs/1509.06461.