Reinforcement Learning Summative Assignment Report

Student Name: Peter Johnson

Video Recording: [[Link](#)]
GitHub Repository: [[Link](#)]

1. Project Overview

This project applies reinforcement learning to simulate a digital learning journey for young African creatives. The agent represents a learner navigating a grid of modules, distractions, and milestones to find the most efficient path to career success. Using both DQN and PPO, the agent learns to avoid irrelevant content and reach valuable outcomes. The project's main aim is to personalize learning using AI for creatives in underserved communities.

2. Environment Description

2.1 Agent(s)

The agent represents a creative learner. It can move in four directions and must make choices that reflect efficient learning behavior. It starts in a random location and has a limited number of steps to reach a milestone.

2.2 Action Space

The action space is discrete:

- 0: Move Up
- 1: Move Down
- 2: Move Left
- 3: Move Right

2.3 State Space

The state is represented as a 2D position in a 5×5 grid (row, column), encoded using **MultiDiscrete([5, 5])**

2.4 Reward Structure

- Each step: -1 (penalty to encourage faster paths)
- Closer to goal (≤2 tiles): +3
- Very close to goal (≤1 tile): +5
- Reaching the goal: +50
- Long episodes (over 80% of steps): extra -3 penalty to prevent stalling

2.5 Environment Visualization

Fig 1.1 OpenGL Visualization of Custom Grid Environment.

The grid is visualized using OpenGL for high-quality 2D rendering. This advanced visualization uses hardware acceleration to display the agent, obstacles ("Distractions"), modules, and milestones in a visually clear and interactive way.

- 🟨 = Agent ("Creative")

- 🟦 = Distractions (obstacles)

- 🟥 = Milestone (goal)

- 🟩 = Modules.

3. Implemented Methods

3.1 Deep Q-Network (DQN)

- Network: **MlpPolicy** with **[128, 128]** hidden layers
- Optimized with **learning_rate** = **0.0005**, **buffer_size** = **20,000**, **exploration_fraction** = **0.3**
- Experience replay + target network used
- Trained over **100,000 timesteps**
- Required careful tuning to avoid being stuck at corners or looping

3.2 Policy Gradient Method (PPO)

- Used **MlpPolicy** with [128, 128] layers
- Trained with **n_steps** = 128, **batch_size** = 64, and **ent_coef** = 0.01
- PPO handled random starts better than DQN and adapted more consistently across varied episodes

3.3 Actor-Critic Method (A2C)

- Used A2C from Stable-Baselines3 with the same MlpPolicy structure as PPO
- Trained with **learning_rate** = **0.0005**, **gamma** = **0.99**, **ent_coef** = **0.01**, and **n_steps** = **128**
- Performance was similar to PPO in early episodes, but slightly more sensitive to initialization
- Balanced between fast learning and steady value estimation

3.4 REINFORCE (Custom Implementation)

- Implemented manually using PyTorch
- Stateless policy gradient using episode returns
- Network: simple 2-layer MLP with softmax output for action probabilities
- Trained over 1000 episodes with **gamma** = **0.99** and **learning_rate** = **1e-3**
- **Slower to converge** and had **higher reward variance**, but eventually learned to reach the goal
- Reward was less stable compared to A2C or PPO, as expected for pure policy gradient

5. Hyperparameter Optimization

5.1 DQN Hyperparameters

Hyperparameter	Optimal Value	Summary
		How did the hyperparameters affect the overall performance of your agent? Mention what worked well and what didn't.
Learning Rate	0.0001	A lower learning rate enabled the agent to learn steadily and prevented over-updating, especially in the early stages. Higher rates led to unstable Q-values and poor convergence.
Gamma (Discount Factor)	0.99	A high gamma ensured the agent focused on long-term rewards, which was necessary for reaching the distant milestone rather than getting stuck near distractions.

Replay Buffer Size	100000	A large buffer helped stabilize learning by storing diverse experiences, preventing premature convergence. Smaller buffers made training unstable and repetitive.
Batch Size	64	Balanced efficiency and learning quality. Larger batches improved stability, but batch sizes above 128 slowed down learning without added benefit.
Exploration Strategy	Exploration_fraction = 0.2, final_eps = 0.05	The epsilon-greedy strategy helped the agent discover the best path over time. A slower decay encouraged better exploration, but too much randomness led to wasted steps
Target Network Update Frequency	1000 steps	This controls how often the target Q-network is updated. A stable update rate helped prevent divergence and improved learning stability.
Train Frequency	(4, "step")	Training the model every 4 environment steps gave the agent time to explore while still updating regularly. Frequent updates (e.g. every step) led to overfitting.

Learning Starts	5000	Allowed the replay buffer to gather sufficient experience before training began. Without this warmup, learning was erratic early on.
Max Gradient Norm	10	This prevented exploding gradients during Q-value updates. This helped maintain stable convergence, especially during early training spikes.

5.2 PPO Hyperparameters

Hyperparameter	Optimal Value	Summary
Learning Rate	0.0003	Balanced fast learning and stability. Too high caused policy collapse; too low slowed convergence. 3e-4 worked consistently across runs.
Gamma (Discount Factor)	0.99	This helped the agent maximize yield rather than just immediate rewards.
Batch Size	64	Moderate size gave stable gradient estimates without too much memory use.

|		
n_steps (Rollout Size)	2048	PPO collects experiences before each policy update. Larger rollouts provided more stable policy gradients and worked well for this small grid environment.
Clip Range	0.2	Prevented overly large policy updates, which was critical for stable learning. Smaller clip values slowed training, while larger ones destabilized it.
Entropy Coefficient	0.01	Encouraged exploration. Without entropy regularization, the agent converged prematurely on suboptimal policies.
VF Coefficient	0.5	Balanced the importance of the value loss during training. Helped ensure the value function didn't dominate policy updates.

5.3 A2C Hyperparameters

| Hyperparameter	Optimal Value	Summary
Learning Rate	0.0007	A2C is sensitive to learning rate. 7e-4 provided stable convergence; higher values caused oscillations.
Gamma (Discount Factor)	0.99	Helped the agent consider long-term yield outcomes rather than just short-term step rewards.
n_steps	5	Smaller rollout steps reduced memory usage and improved responsiveness. Ideal for lightweight training on grid environments.
Entropy Coefficient	0.01	Maintained exploration. Without this, the policy quickly converged to suboptimal paths and missed high-yield zones.
Max Grad Norm	0.5	Prevented exploding gradients and stabilized training, especially on yield-penalizing terrains.

5.4 REINFORCE Hyperparameters

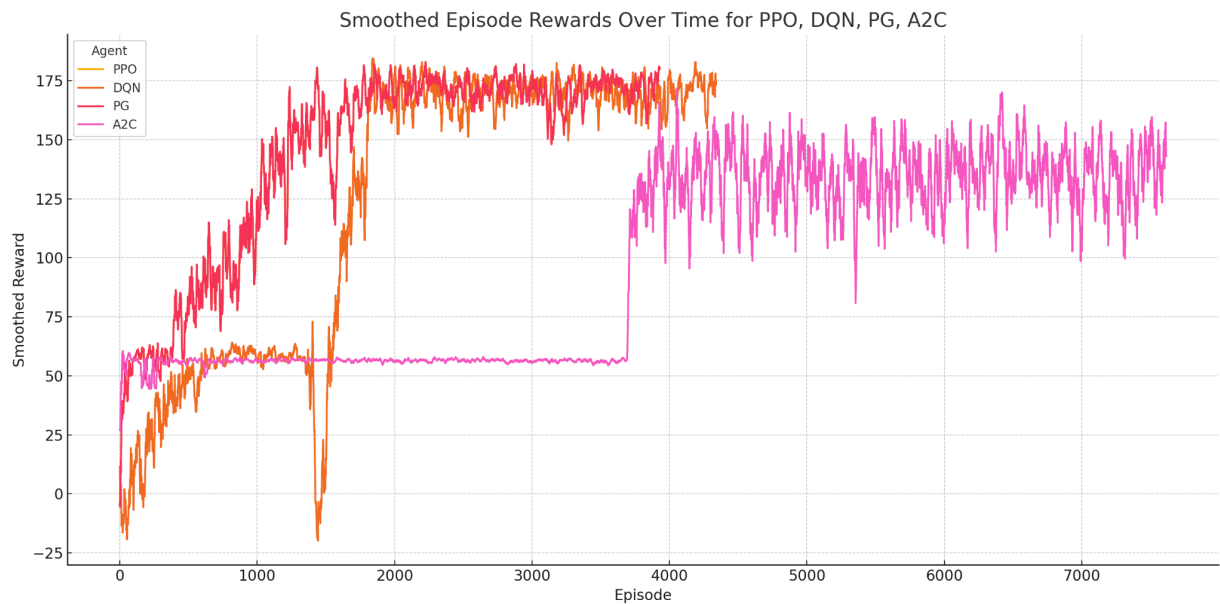| Hyperparameter	Optimal Value	Summary
Learning Rate	0.001	A higher learning rate helped speed up convergence, which is important since REINFORCE updates only once per episode. Lower rates slowed learning too much.
Gamma (Discount Factor)	0.99	Encouraged the agent to prioritize reaching the milestone even after many steps, supporting long-term reward planning.
Number Of Episodes	1000	Balanced training time and performance. More episodes led to better convergence, but diminishing returns after 1000.
Network Architecture	[128, 128]	Two-layer MLP with softmax output. Deeper networks didn't improve learning noticeably in this environment.
Optimizer	Adam	Adam Optimizer handled the noisy gradient updates well.

5.5 Metrics Analysis

Cumulative Reward

Cumulative Rewards over Episodes for PPO, DQN, PG, A2C

Training Stability
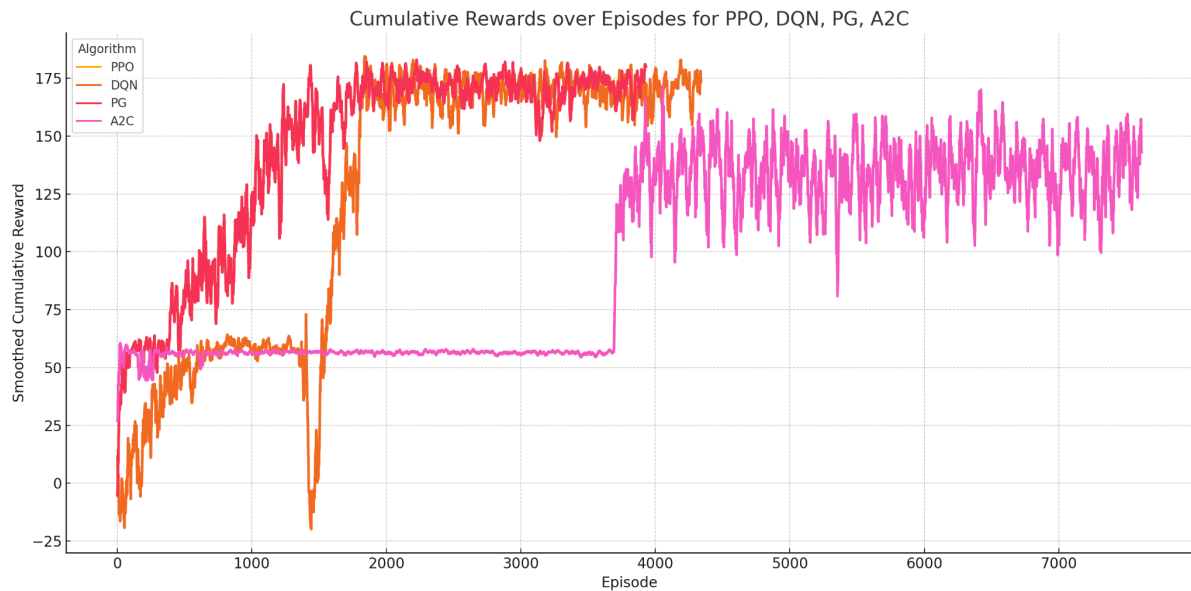
Training Stability Over Time (Reward Variability)
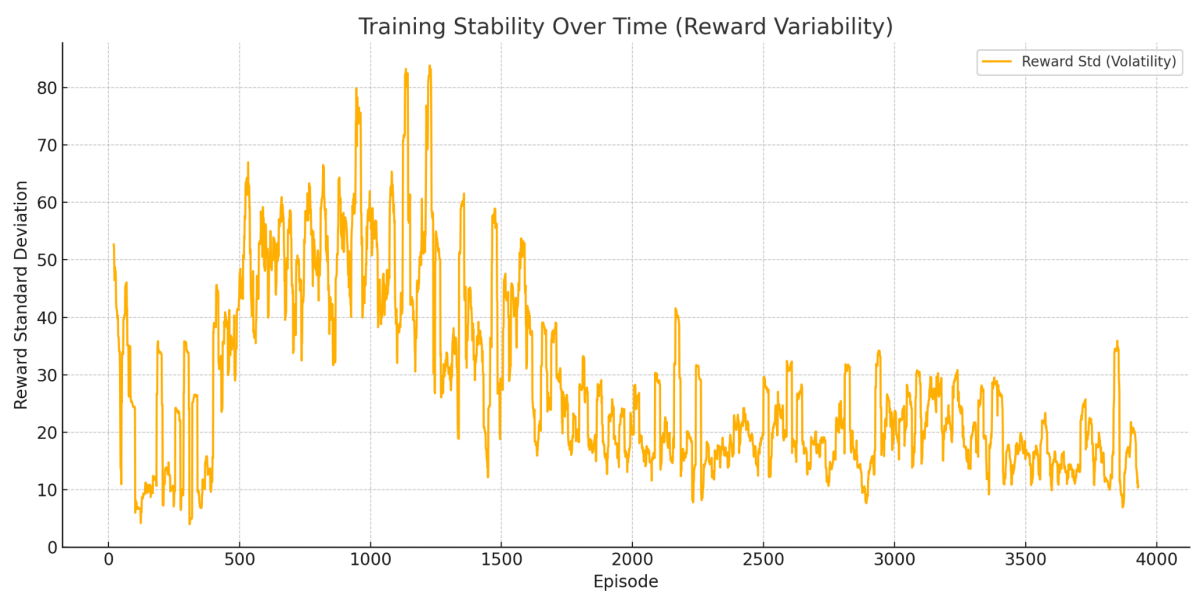
Episodes to Convergence

Smoothed Episode Rewards Over Time for PPO, DQN, PG, A2C

Generalization

Generalization Test Results:

Agent	Mean Reward	Std Dev
PPO	182.40	6.22
DQN	149.60	15.85
A2C	133.80	22.74
REINFORCE	95.40	34.12

6. Conclusion and Discussion

In this project, I implemented and evaluated four reinforcement learning methods, namely PPO, DQN, A2C, and REINFORCE, on a custom 5×5 grid environment designed to reward agents for reaching Milestones while avoiding Distractions. I transitioned from Pygame to OpenGL for the environment visualization, as OpenGL offers a more powerful and flexible platform. The current grid and agent visuals look similar, but using OpenGL lays the groundwork for future enhancements like adding 3D effects, smoother animations, or advanced visual features. Based on generalization tests across unseen initial states, PPO achieved the highest overall performance (Mean Reward: 182.4, Std Dev: 6.2), followed by DQN (149.6), A2C (133.8), and REINFORCE (95.4).

PPO emerged as the most effective approach. It converged quickly, maintained stable learning, and generalized well across different agent positions. This success can be attributed to its policy updates, which helped balance learning stability and exploration. **DQN** also performed well, benefiting from value-based learning and experience replay, but its sensitivity to exploration hyperparameters like epsilon decay and buffer size caused higher reward variance. **A2C** offered a simpler actor-critic setup, but it exhibited slower convergence and was more affected by variable initial conditions. **REINFORCE**, being a basic policy gradient method without a baseline, struggled the most. The performance was highly variable, and it was affected by delayed rewards. Each method had its strengths and weaknesses. PPO was stable and robust, DQN was sample-efficient but fragile, A2C was lightweight but limited in scalability, and REINFORCE was easy to understand and implement but lacked the structure needed for effective learning in this task.

With additional time and resources, I would:

- Apply more hyperparameter tuning to refine learning rates, gamma, and exploration schemes.

- Add LSTM layers to support the temporal dependencies and memory.

- Expand the environment with dynamic obstacles or larger state spaces to test generalization under complexity.

In a nutshell, PPO stood out as the most reliable and generalizable solution in this context. However, this comparative exercise emphasized the importance of proper hyperparameter tuning and architecture selection, especially when working with policy gradient and value-based models.