Assessing Reinforcement Learning Models in Exploration-Driven Minigrid Enviornments

CS 5756 - Robot Learning (Spring 2024) - Group 22

May 11, 2024

Anthony Coffin-Schmitt (awc93)

Huan Pham (tp397)

Hansal Shah (hms262)

1 Introduction

This project investigates the performance of three deep reinforcement learning (RL) algorithms on Mingrid[1] gym environments. These environments are sparse and randomized which favors the algorithms that explore. The rewards in the environment are sparse, meaning the agent only receives a reward only if it ever reaches the goal, else it receives a 0 reward.

We implemented the following three algorithms:

1. Proximal Policy Optimization (PPO)[6]
2. Advantage Actor Critic (A2C)[4]
3. Deep Q-Learning (DQN)[3]

For all the three algorithms, we trained three different versions of each one of them, in three different Minigrid environments, resulting in a total of 27 trained models.

Experimental environments:

1. Empty Room (`MiniGrid-Empty-16x16-v0`)
2. Empty Room with Randomized Goal (`Custom environment`)
3. Four Rooms (`MiniGrid-FourRooms-v0`)

Experimental model versions:

1. Vanilla Stablebaseline Models
2. Stablebaselines Models with action restriction custom wrapper
3. Stablebaselines models with action and reward shaping custom wrappers

Our project goal is to implement the models and evaluate the performance of each of the algorithms using mean reward and mean length of episode metrics. We also will discussion the strengths and shortcoming of the different algorithms and offer insights into the motivations of the algorithms in the sparse environments, such as Minigrid.

2 Problem

2.1 Minigrid Environment Space

The Mingrid environments are 2D grids with different challenges for a triangular agent in the world that is trying to solve the task. All the Minigrid environments have the following things in common:

1. **Action Space**
   The action space is discrete with 7 possible actions as listed in Table 1.

Number	Name	Action
0	left	Turn left
1	right	Turn right
2	forward	Move forward
3	pickup	Pick object
4	drop	Drop object
5	toggle	Switch objects
6	done	Done completing task

Table 1: Action Space Details

2. **Observation Space**
   All the agents in the environment observe a 7×7 field of view that is described by the following data structure: `Box(0, 255, (7, 7, 3), uint8)`. Each tile is encoded as a 3 dimensional tuple: `(OBJECT_IDX, COLOR_IDX, STATE)`. Along with this, we also observe the direction in which the agent is facing as a discrete integer: `Discrete(7)`
3. **Rewards**
   The agent receives a sparse reward from the environment once it reaches its goal state or successfully completes its task. The rewards for the agent is calculated as follows by default:

$$1 - 0.9 * (\texttt{step\_count}/\texttt{max\_steps})$$

2.2 Room Environments

Listed below are the three Minigrid environments that we trained our agents:

1. **Empty Room** (`MiniGrid-Empty-16x16-v0`): This environment is an empty room with a width and height of 16 cells. The goal of the agent is to reach the green square, which provides a sparse reward. The goal is always located in the lower right corner of the grid. A small penalty is subtracted for the number of steps to reach the goal.
2. **Empty Random Room**: This is same as the empty room except we engineered the environment to randomly position the goal state every time the environment is reset.
3. **Four Rooms** (`MiniGrid-FourRooms-v0`): The agent navigates in a maze composed of four rooms interconnected by 4 gaps in the walls. To obtain a reward, the agent must reach the green goal square. Both the agent and the goal square are randomly placed in any of the four rooms.

The main challenge for the agent in these environments is sparse rewards. They receive feedback only infrequently (at the end of the episode / rollout) which makes it difficult for the agent to determine which actions were beneficial and which were not, as the reward signal is not sufficiently informative to guide learning through the episode. Sparse rewards also hinder exploration because the agents might not encounter a reward at all if they don't explore a efficiently, leading to slow or stagnant learning.

2.3 Model Selection

For this problem, we hypothesize that **Proximal Policy Optimization** (PPO) algorithm will outperform both the **Deep Q-Network** (DQN) and the **Advantage Actor-Critic** (A2C) in terms of cumulative rewards and stability of training.

We attribute this superiority to the following:

- PPO's ability to balance exploration and exploitation through a more flexible and adaptive policy update mechanism, which is crucial in environments with high-dimensional state spaces and complex decision-making processes.

- In contrast, DQN might perform moderately due to its efficient learning in discrete action spaces but could suffer from overestimation and convergence issues.

- A2C is expected to show a faster initial learning curve due to its on-policy advantage learning but might struggle with higher variance in policy updates compared to PPO, affecting its overall performance in environments with diverse and dynamic challenges.

3 Approach

3.1 Model Implementation

We first approached the problem by implementing 'vanilla' versions of the algorithms without any hyperparamter tuning or wrappers. We wanted to gain a baseline of how the models performed with no interventions. To remove potential bias from our algorithm implementation, we utilized Stable Baselines3 [5] package for the algorithms. Note in all three algorithms, we used MLP policy (2 layers of 64) and provided flattening wrapper to flatten observation before training. Our evaluation of the models was done by taking the mean reward from 50 random episodes.

We had originally intended to implement Q-Learning [7], however we found that Q-Learning was unable to handle the large state space in Minigrid so we modified to instead implement A2C. Each team member focused on implementing a specific algorithm and assessing its performance. We eventually decided to implement a RL pipeline to train and evaluate the models in a more uniform manner.

The models were trained for 200,000 time steps in the Empty Room and the Empty Room with Randomized Goal and for 1 million time steps in the Four Rooms environment. The increase in time steps was necessary to accommodate for the increased difficulty of solving the more complex grid. The models were implemented on a vectorized environment with $n = 4$. Training and evaluation metric results were stored in TensorBoard[2] and figures were generated from the datasets.

3.2 Wrapper Implementations

A low hanging fruit improvement was to apply an action wrapper. In the default environment the agents could choose seven actions to perform, but our environments only needed three, specifically turn left, turn right, and move forward. Removing the useless actions greatly decreased the action space. So our next iteration in the models was to generate 'Action' models that had an action space wrapper applied. These models were all retrained in the different environments

As a final approach, we applied reward shaping algorithms to the agents. We tried many different variations and found some actually hindered the agents performance and others gave slight improvements. As new reward shaping features were applied, we found we may have to add additional features for correction. For instance, in the FourRoom environment, a reward feature that encouraged exploration seem to primarily make the agent only explore the current room and not leave, so a new feature added a cost for 'over' exploring a room.

4 Results

The following Figures show both the mean episode reward and the mean episode length in each of the three main environments. The Reward-shaping models results are plotted in different figures from the Vanilla and Action models due to being on different reward scales and thus not as easy to compare or visualize.

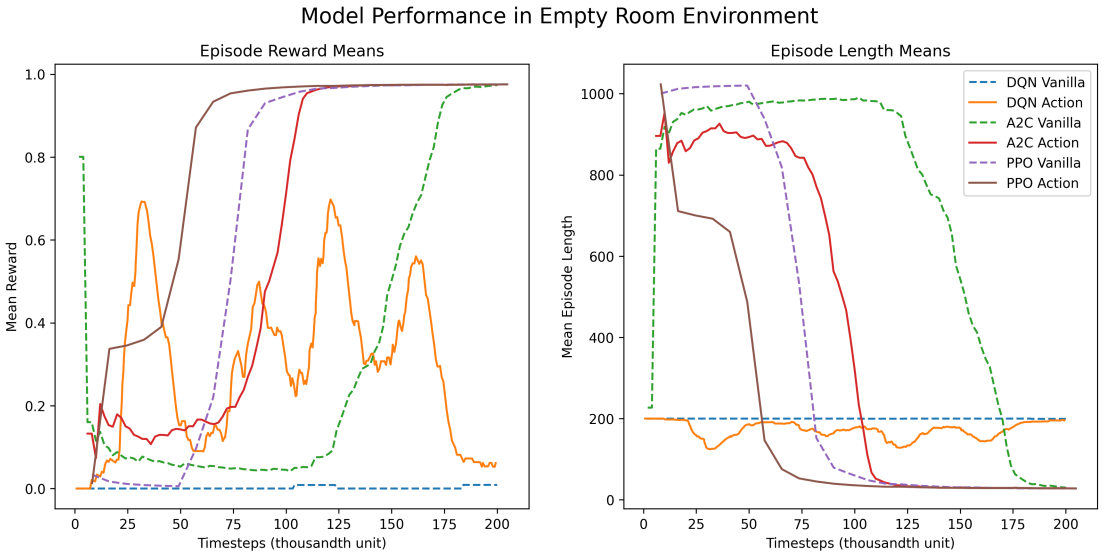

Figure 1: Comparison of how models compared under the Empty Room 16x16 environment with set reward location. Both the mean episode rewards and mean episode length, i.e. steps taken by agent, are shown.

PPO and A2C outperform DQN in the empty room environment. This highlights the importance of on-policy learning for better performance in the sparse environments. Other thing to notice here is that DQN is very unstable and it finds many sub optimal policy and never converges to one like PPO and A2C.

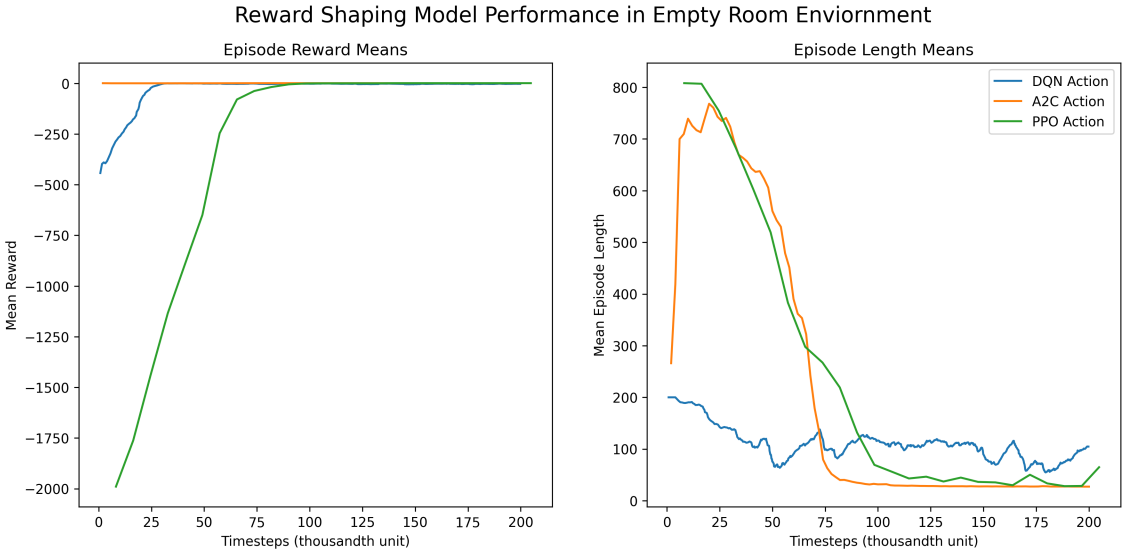

Figure 2: This figure illustrates the difference reward shaping made in the Empty Room environment, as both the A2C and DQN models were able to converge to better performing policies.

This discusses the reward shaping in the empty room environment. The graphs indicate that it helps all the algorithms to converge faster and to the same optimal policy. Reward shaping helps overcome the sparse reward curse of the environment which helps DQN to be stabilized and eventually find an optimal policy.

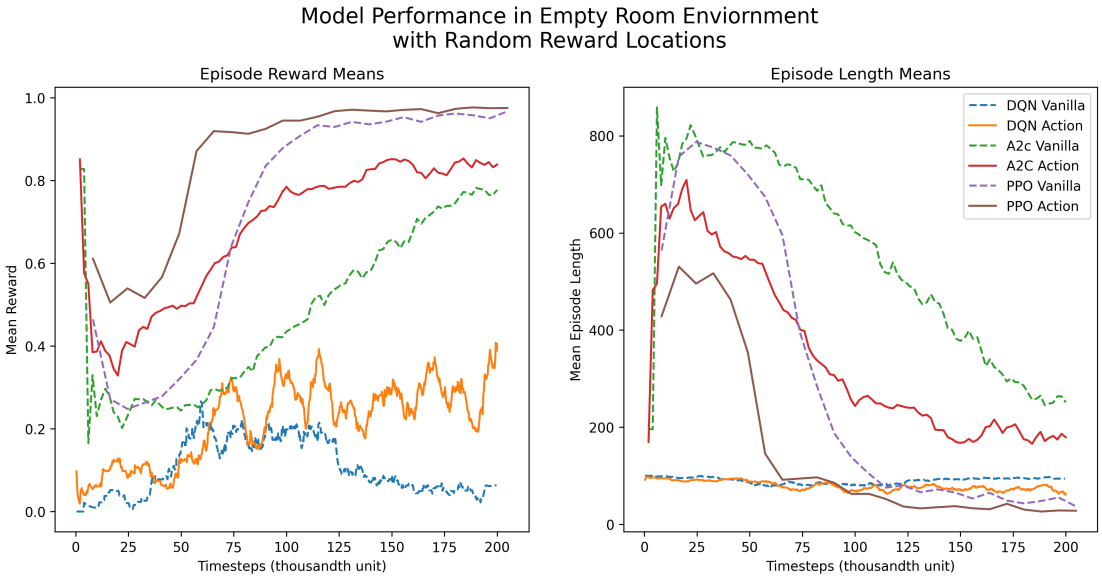

Figure 3: Comparison of how models compared under the Empty Room 16x16 environment that has a random reward generation for each episode. Both the mean episode rewards and mean episode length, i.e. steps taken by agent, are shown.

When the models were trained in the empty random room environment, we can see the general trend of slower convergence for all the algorithms. Also, none of them reach the same performance as they do in the simple empty room environment. DQN again shows signs of instability and reach sub optimal results.

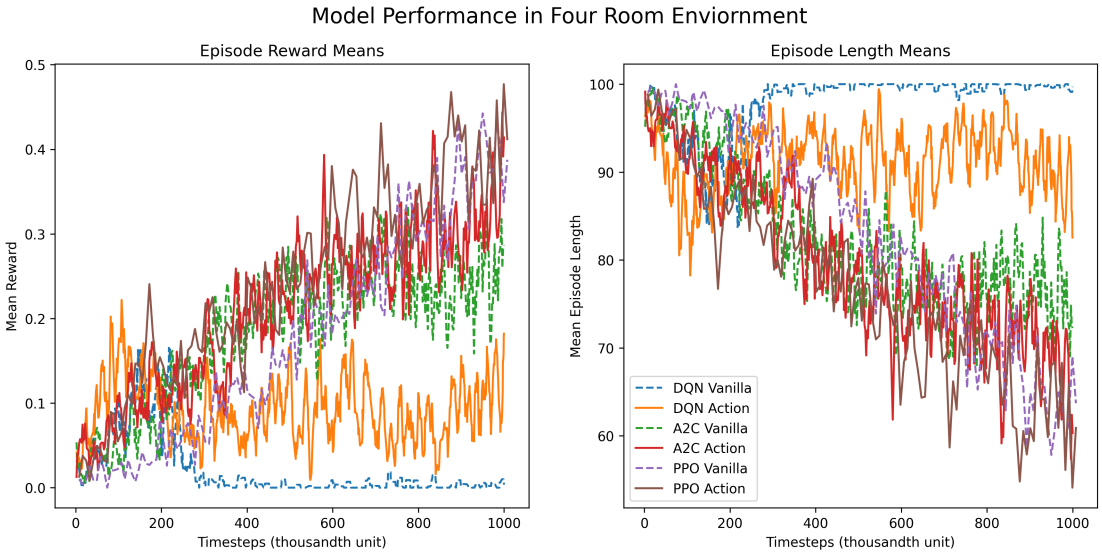

Figure 4: Comparison of how models performed in the Four Room environment which was the most challenging. Both the mean episode rewards and mean episode length, i.e. steps taken by agent, are shown. The increase in model performance variance is likely due the random placement of the agent and reward can drastically change the difficulty of that episode.

It shows the learning progress for all the algorithms in the Four Rooms environment. There is a lot of variance in the training stability for all the algorithms which we attribute to the randomness in the gaps in the walls, the starting position and direction of the agent, and the goal state. The performance caps at 0.4 for all of them.

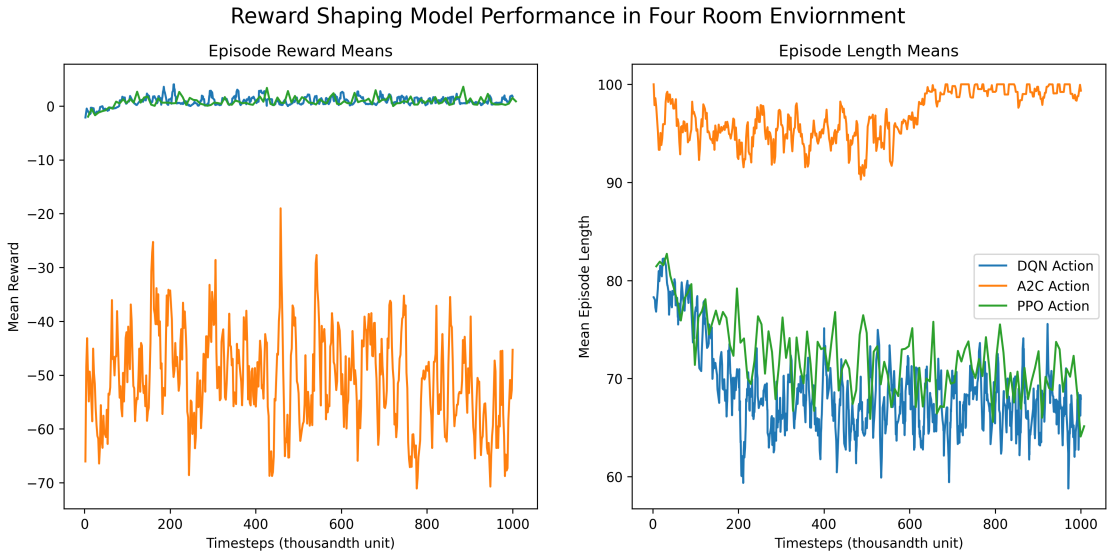

Figure 5: This figure shows results from a reward shaping wrapping applied to the different models in the Four Rooms environment. There were variations in the reward shaping features applied which also explains the negative score. This also means that a direct comparisons is not as informative, but we can still observe the models independently.

Links

1.	Deep Q-Networks (DQN)	arXiv Paper	
2.	Proximal Policy Optimization	arXiv Paper	
4.	Minigrid	Documentation	Github
5.	Stable Baselines3	Documentation	Github
6.	Video Spotlight Talk	Video	

References

[1] Maxime Chevalier-Boisvert et al. "Minigrid & Miniworld: Modular & Customizable Reinforcement Learning Environments for Goal-Oriented Tasks". In: *CoRR* abs/2306.13831 (2023).

[2] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[3] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: `1312.5602 [cs.LG]`.

[4] Volodymyr Mnih et al. *Asynchronous Methods for Deep Reinforcement Learning*. 2016. arXiv: `1602.01783 [cs.LG]`.

[5] Antonin Raffin et al. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: `http://jmlr.org/papers/v22/20-1364.html`.

[6] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: `1707.06347 [cs.LG]`.

[7] Christopher J. C. H. Watkins and Peter Dayan. "Q-learning". In: *Machine Learning* 8.3 (May 1992), pp. 279–292. ISSN: 1573-0565. DOI: `10.1007/BF00992698`. URL: `https://doi.org/10.1007/BF00992698`.