Reinforcement Learning - Final Project

Stav Cohen & Avner Duchovni

March 2025

1 Introduction

In this project, we tackled the problem of training an autonomous agent to navigate through multiple rooms in the MiniGrid multiroom environment using deep reinforcement learning. The room observability was partial, which means the agent doesn't see the whole environment, and we had to solve the large room which contains 6 large rooms.

2 Methodology

2.1 Algorithm Overview

Our project primarily explored two reinforcement learning approaches based on an abstract base algorithm class here:

- **Value based algorithm - Deep Q-Network (DQN):**
  training algorithm reference
  evaluate agent reference
  Utilizes the Bellman equation to estimate action values:

  $$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$$

  While effective in small environments, standard DQN suffered from instability, prompting us to adopt a Double Dueling DQN architecture.

  We used a memory buffer to store the experiences: memory reference

- **Policy based algorithm - Proximal Policy Optimization (PPO) over Actor Critic (AC) Neural Netowrk Model:**
  training algorithm reference
  evaluate agent reference
  A policy gradient method that optimizes policy updates while ensuring a controlled update step using a clipped surrogate objective and using the following equation for policy update:

  $$L(\theta) = E[min(r(\theta)A, clip(r(\theta), 1-\epsilon, 1+\epsilon)A)]$$

  . PPO demonstrated better stability and efficiency in learning complex policies compared to DQN.

In order to get started, we found the project in [3] and got some initial aid based on this project. However, quickly we modified their implementation and made our own version that matches the requirements of this specific project

2.2 Pre-Processing

Reference here
To optimize the agent's perception of the environment, we extracted only the green channel from the image observation, as it contains sufficient information (goal location is green) and converted the observation into a torch array.

2.3 Reward Shaping

Reference here
Reward shaping played a crucial role in improving the agent's learning efficiency. We tried many reward shaping methods, including 3 versions used:

- **Reward Shaping 2**: A basic reward shaping function that doesn't aware to the location of the agent. It gives a reward for moving forward with doing progress and a penalty when hitting a wall. Using the same approach, it gives a reward for opening a door in case of toggle action and a penalty when a toggle action used with no reason. In addition, during turning, it gives a penalty in case of a redundant actions. A great reward for a victory is also given (Reaching goal). This method was the only method which worked in the DQN algorithm (Small room) as can be shown here and had also a partial succesfull in PPO (Small room) as shown here

- **Reward Shaping 5**: An improvement of the original reward shaping function. It also remembers the locations of the agents, and gives an exploration reward when going to a new location. In addition it gives a penalty for "stucking" in the same action more than 3 times. It had not any success in the DQN algorithm as shown here but it had a much great success in the PPO algorithm small room as shown here. However, in the middle room it failed to success for great percentages, and unfortunately it almost didn't open doors and stucked into looping inside the same room as shown here

- **Reward Shaping 3**: Based on Reward shaping 5, we increased the reward of opening a door and decreased the reward of making forward, but the main improvement was adding a penalty of visiting the same location over 3 times. It let us succeed both the middle room (here) and the large room in PPO here

2.4 Hyperparameters

We extensively tuned hyperparameters to optimize performance. The final values used per algorithm can be shown in the notebook:
DQN Small Room: Notebook; **DQN Medium Room**: Notebook
DQN Large Room: Notebook; **PPO Small Room**: Notebook
PPO Medium Room: Notebook; **PPO Large Room**: Notebook
We've conducted further hyperparameter tuning, which can be found under the

Results section. These include testing DQN with different values for the Decay, Learning Rate, and Target Policy update. In PPO we've experimented with values for General Advantage, Clipping and PPO Epochs (See results for links and references)
In DQN we found that using high exploration rate at start and a slow decay gives the best results, and also updating the target network every 50 steps with a batch size of 64 which is a medium-size batch
In PPO, we found that using some entropy co-efficient for exploration is critical in order to not be fixed to the same patterns, and use a medium clip-ebs is also vital to keep medium updates but not too small or large. Because we didn't use the LTSM layer which stores memory, we increased the number of PPO Epochs for each update to make a stable trainning and reduced the number of re-currences to avoid overfitting. Also we found out that seperate the number of frames per update is also critical so every update will include 20-40 episodes in order to learn multiple patterns

2.5 Deep Learning Architecture

We employed a convolutional neural network (CNN) to process image observations both in DQN and in PPO over AC. Both were implemented in PyTorch. The architecture consisted of:

- Double Dueling DQN network can be shown here
- Actor Critic network can be shown here
- In both networks we use three convolutional layers which gets the observability tensor as input and Relu function between them
- In DuelingDQN we use also a value stream layer and then an advantage stream layer which outputs the actions to use (There are final 4 output channels, one for each action)
- In AC we use also an Actor layer which outputs the action to use (There are final 4 output channels, one for each action) and acts as the Policy layer and a Critic layer which outputs the value function to use based on evaluating the action to use and acts as the Value layer. The Actor tries to maximize the reward and the Critic tries to correct the policy proposed by the Actor for better choises in the future. There is also an optional layer of LTSM (Long Term Storage Memory) which stores memory of previous observations. However we could not use it due to RAM limits

3 Results

First, we tried to use Deep Q-Networks (DQN), but it resulted in poor learning performance. Consequently, we adopted a double and dueling network architecture, which improved the success rate to approximately 60% in small rooms using the basic reward shaping 2 as shown in figures 1 and 2.

Despite this improvement, scaling to medium and large rooms posed significant challenges due to increased state complexity and longer episode lengths as shown in figures 8, 9, 10 and 11.
To address these issues, we implemented extensive reward shaping and experimented with various techniques to improve the agent's learning capabilities (Reward Shaping 5 and Reward Shaping 3). However, they didn't work well in the DQN small rooms as shown in figures 4, 5, 6 and 7

Ultimately, we implemented Proximal Policy Optimization (PPO), which significantly improved results and allowed us to solve the large room successfully. Our experiments showed that DQN struggled with stability and scalability, achieving a 23% win rate and completing an additional 21% of episodes without winning. Lowering the decay rate (Colab link), lowering the learning rate 34 (Colab link), and increasing the frequency of target network updates 35 (Colab link) initially improved performance but eventually led to a decline, dropping the win rate to around 5% 36

In contrast, PPO demonstrated superior learning efficiency and stability, with a baseline success rate of 36% that steadily increased. Adjusting the general advantage estimate to 0.9 37 resulted in a similar 34.5% win rate, while lowering the clip epsilon to 0.2 38 significantly boosted performance to 51.8% .Increasing the number of PPO epochs to 4, however, slowed the improvement rate, stabilizing at 34.4%. These results highlight the limitations of DQN in complex environments and the advantages of PPO's policy-based approach in reinforcement learning tasks 39 .

By implementing PPO, we observed a significant improvement in performance, ultimately enabling the agent to solve larger rooms successfully.
At first, we succeeded to solve the small room with a great performance as shown in figures 13 14 15 16(Reward Shaping 2), figures 17 18 19 20(Reward Shaping 5)
However, we still got a poor performance in Middle room as shown in figures 21 and 22 using the second version of our Reward Shaping (5), and the most annoying thing is that is started to succeed at the middle and then suddenly stopped and started to do selp loops inside a room as shown in Figures 23 and 24 (Value and policy loss).
After adopting the changes of Reward Shaping 3 we have successfuly solved the Middle room with great performance as shown in figures 25 26 27 and 28 PPO's stability and policy-based approach allowed the agent to generalize better to complex environments.
Then we solved the Large room of PPO using reward shaping 3, but with loading the trained-file of the middle room in order to increase the performance. We trained it twice, once with 1000 episodes (From 50% success to 70%) and in the second time with 500 episodes we got 80% success at the end as can be shown in figures 30, 31, 40 and 41The exploration-driven rewards helped the agent navigate efficiently, while the step penalty successfully encouraged forward movement. Nevertheless, learning stagnated in larger environments with DQN, which led us to explore PPO for better scalability.

4 Comparison of Algorithms

We compared DQN and PPO based on:

- **Performance:** PPO outperformed DQN in terms of success rate and stability, especially in large rooms.
- **Learning Efficiency:** PPO required fewer training steps to converge compared to DQN.
- **Exploration vs. Exploitation:** DQN relies on ϵ-greedy exploration, while PPO inherently balances exploration with policy updates.
- **Partial Observability:** PPO handled partial observability better due to its policy-based nature, whereas DQN struggled with long-term dependencies.

5 Discussion

The key takeaway from this project is the importance of reward shaping in reinforcement learning. By carefully designing rewards to encourage useful behaviors (exploration, goal completion) and discourage inefficient ones (idle actions, unnecessary door toggling), we significantly improved the learning process.

Challenges faced include:

- **Scalability:** As the environment size increased, learning became less effective with DQN but improved significantly with PPO.
- **Sparse Rewards:** Despite our modifications, delayed rewards made it difficult for the agent to form effective strategies.
- **Exploration Balance:** Encouraging exploration without excessive wandering was a challenge.

6 Conclusion

This project demonstrated the effectiveness of Double and Dueling DQN in reinforcement learning tasks with complex state spaces. However, the introduction of PPO significantly improved scalability and performance in larger rooms. Reward shaping played a crucial role in guiding the learning process, particularly in environments with sparse rewards.

Further refinements, such as hierarchical learning and intrinsic motivation techniques, could enhance scalability and generalization.

References

[1] Main Notebook

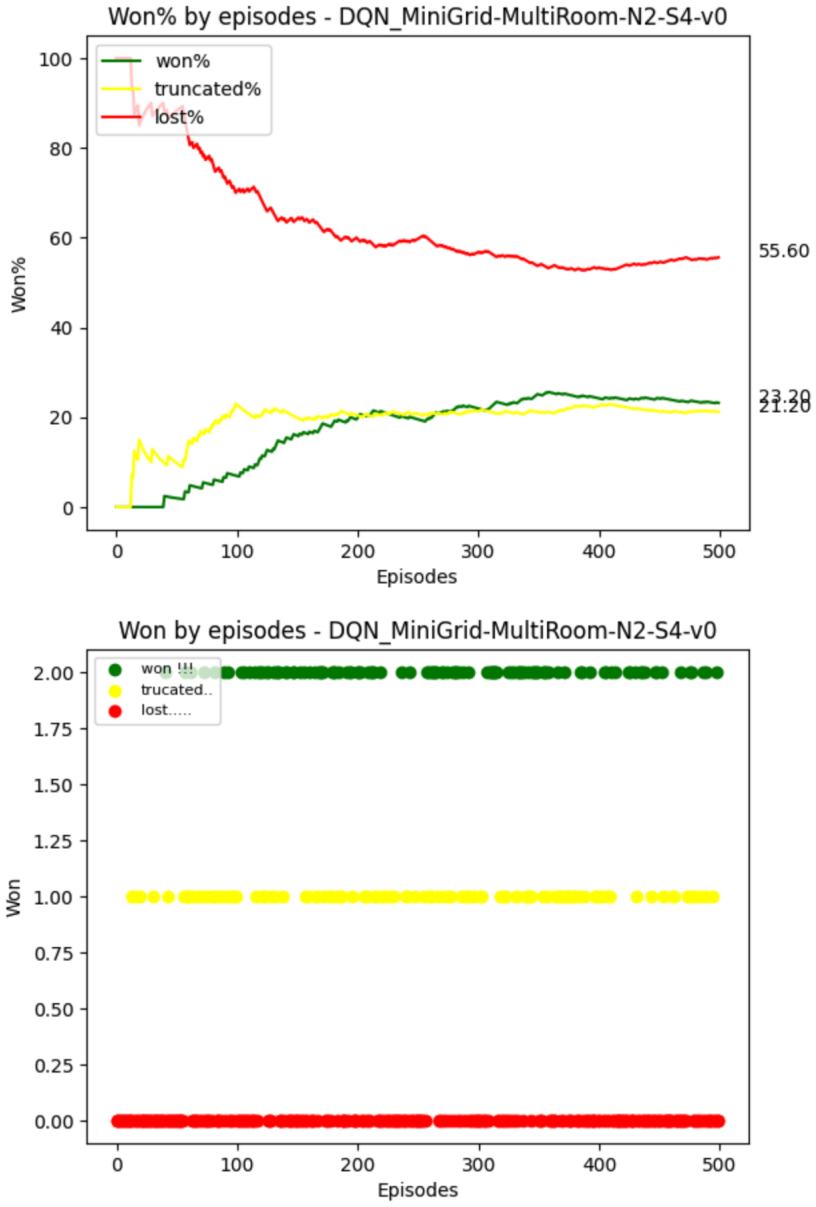
[2] Evaluation Notebook

[3] Minigrid working project in Github

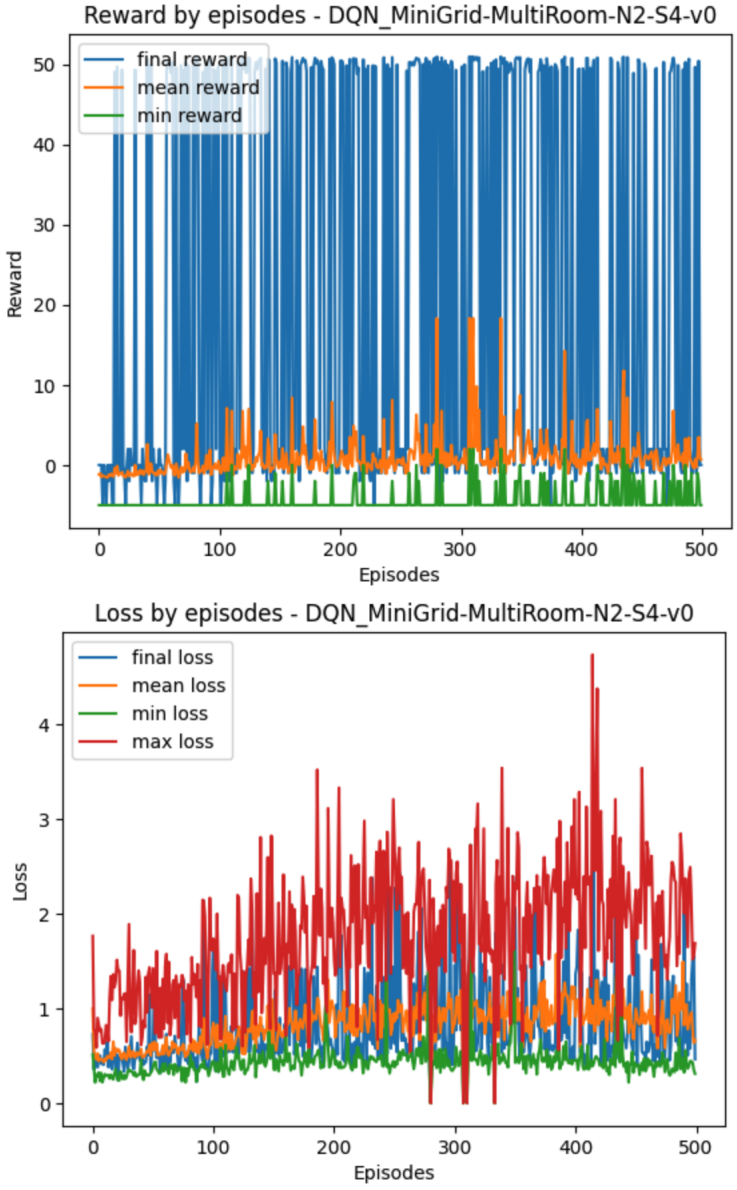
7 Appendix-Graphs

7.1 DQN Small Room - Reward Shaping 2

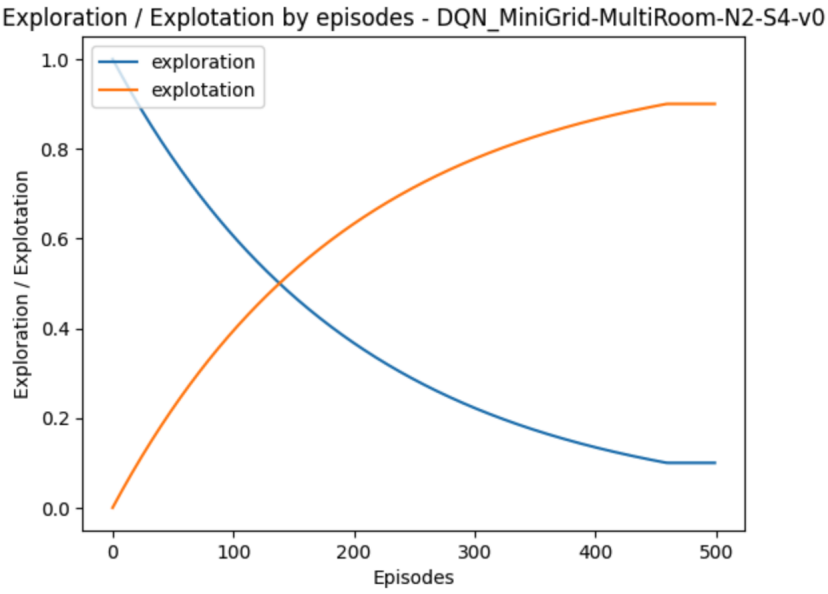
7.1.1 Figure 1 - Wins



7.1.2 Figure 2 - Reward & Loss

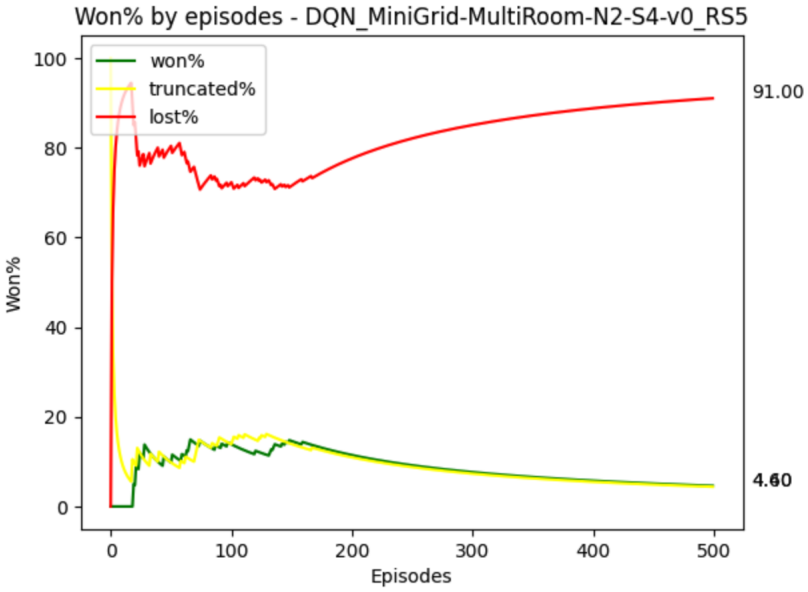


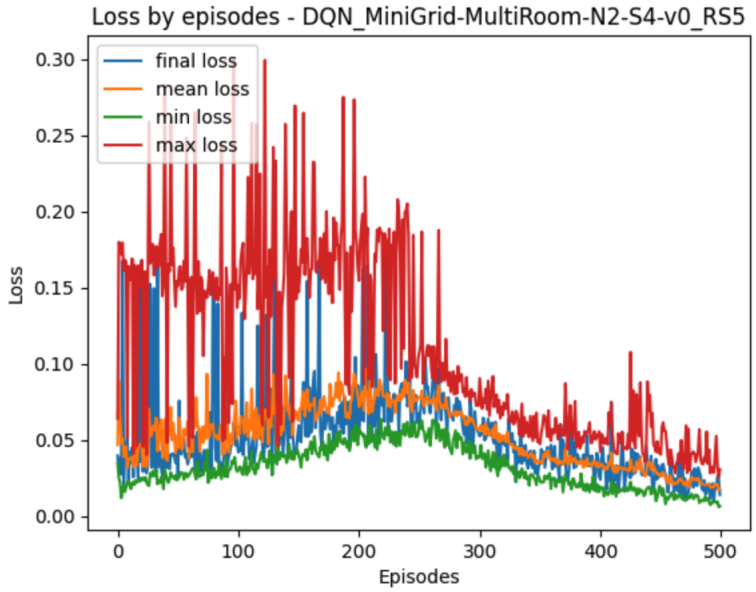
7.1.3 Figure 3 - Exploration / Exploitation



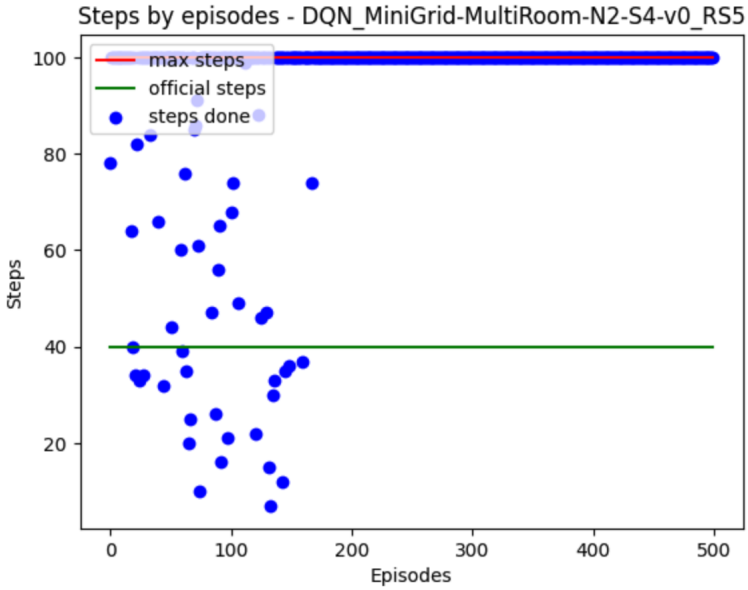
7.2 DQN Small Room - Reward Shaping 5

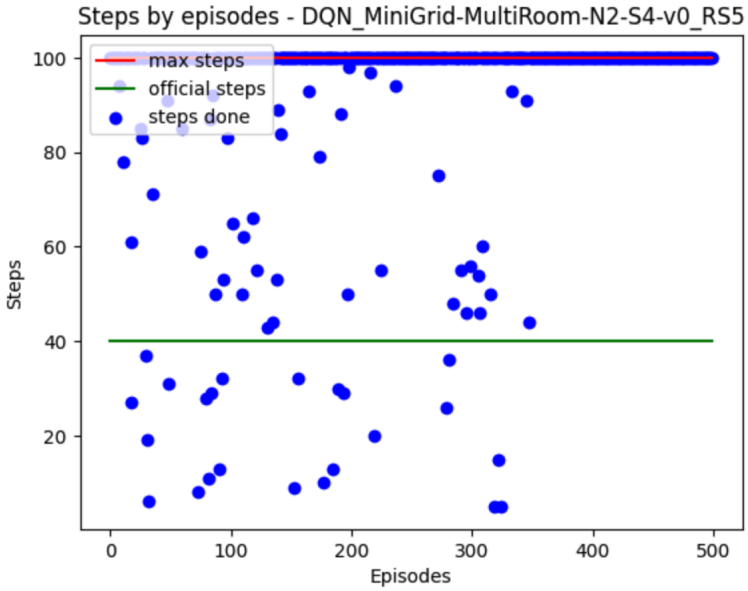
7.2.1 Figure 4 - Wins & Loss





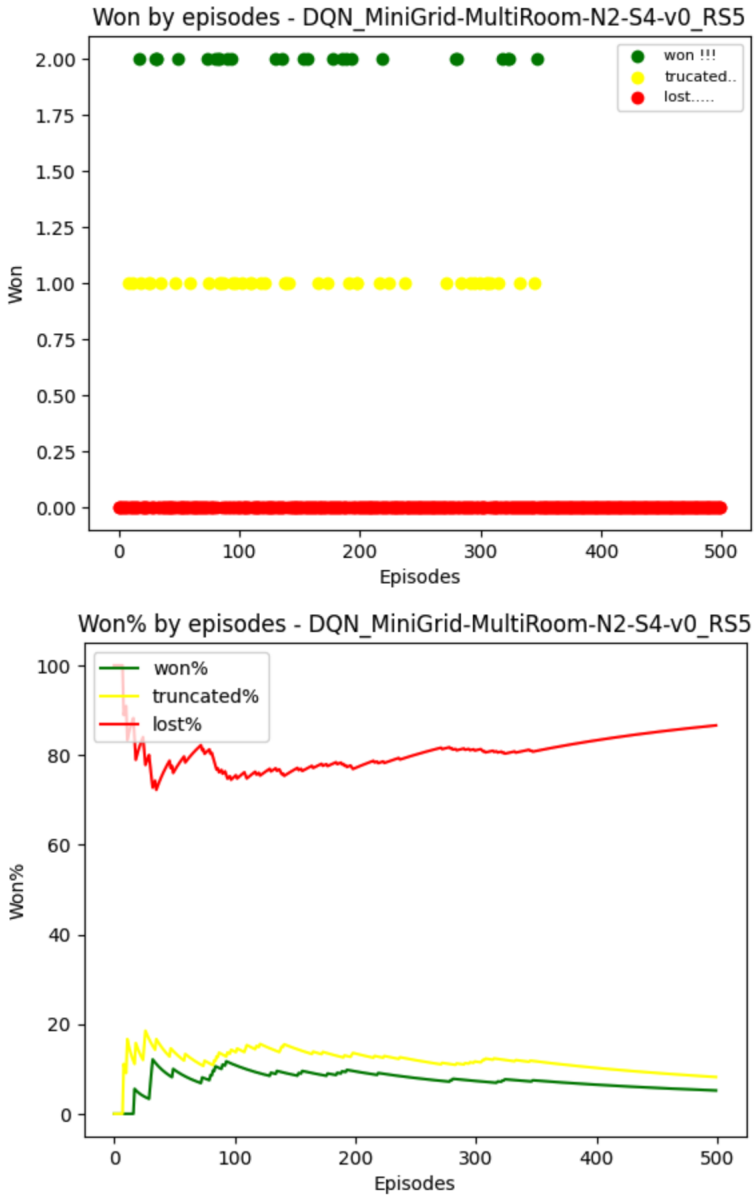
7.2.2 Figure 5 - Steps



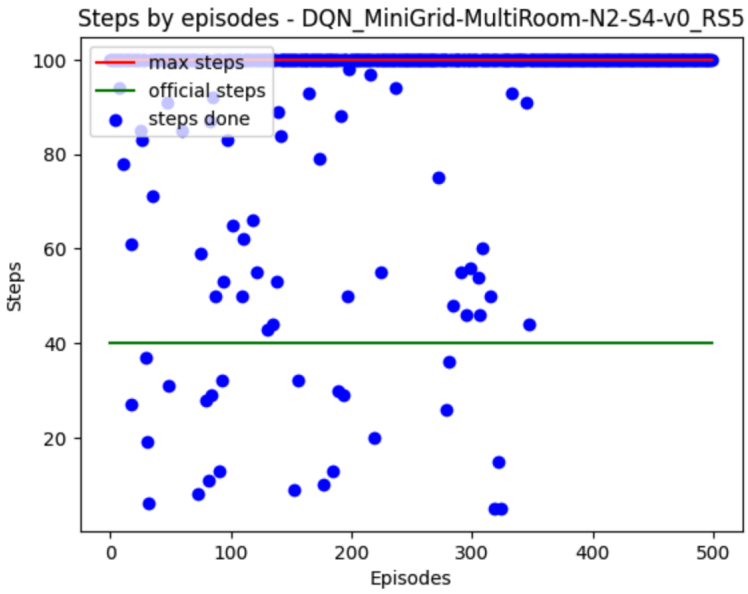


7.3 DQN Small Room - Reward Shaping 3

7.3.1 Figure 6 - Wins

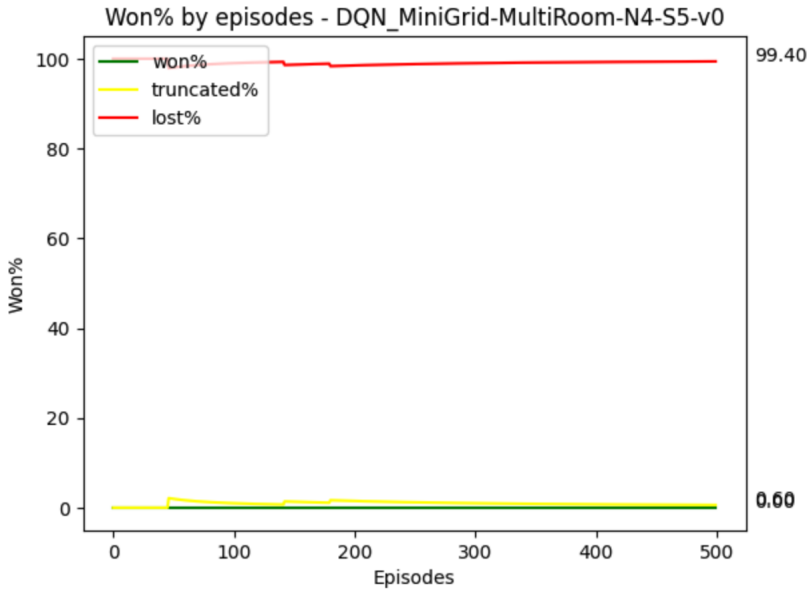


7.3.2 Figure 7 - Steps

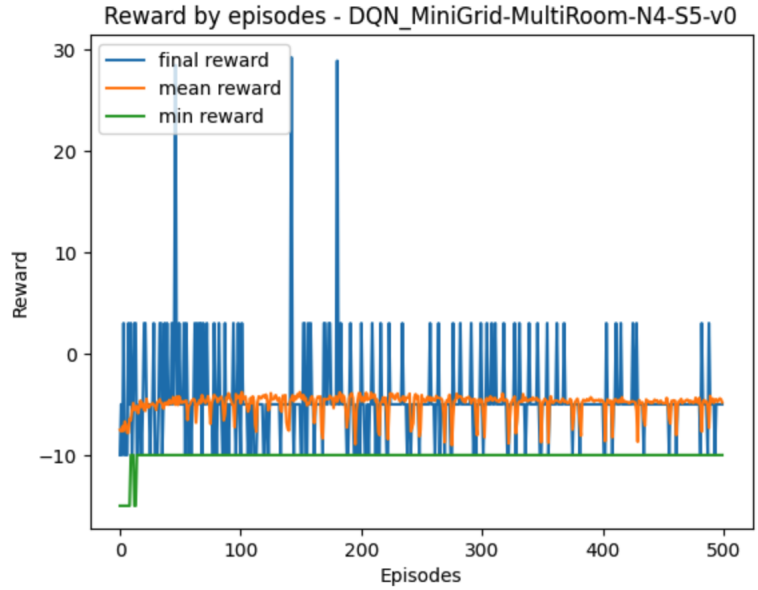


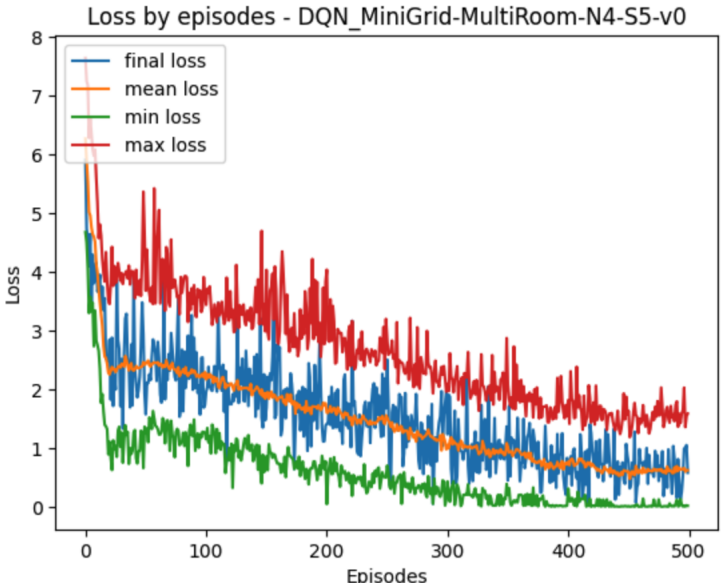
7.4 DQN Medium Room - Reward Shaping 2

7.4.1 Figure 8 - Wins



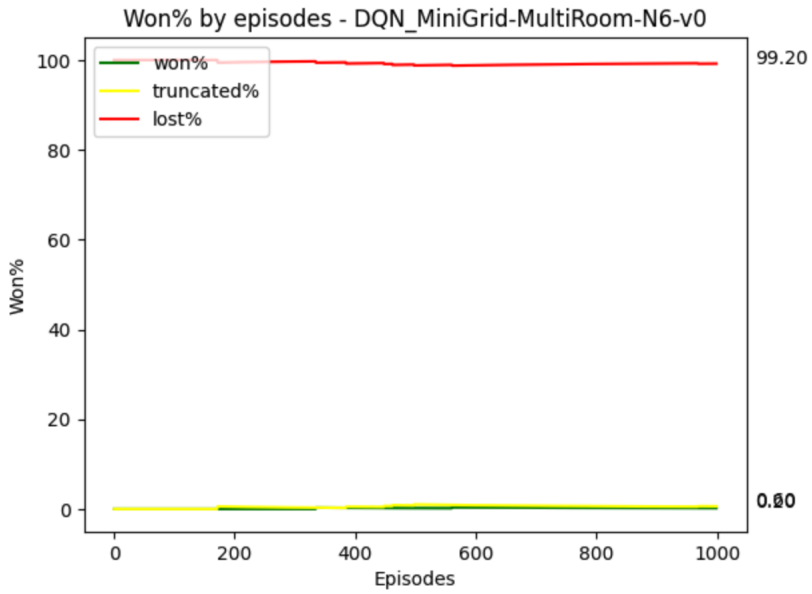
7.4.2 Figure 9 - Reward & Loss



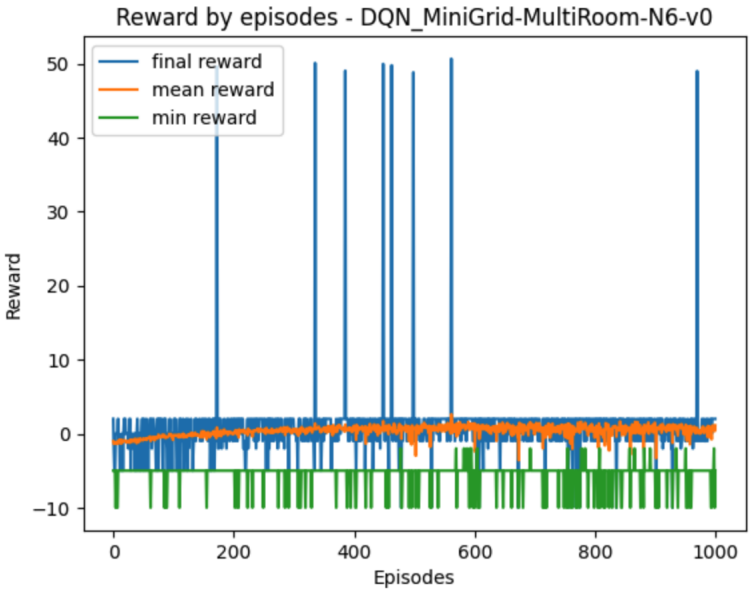


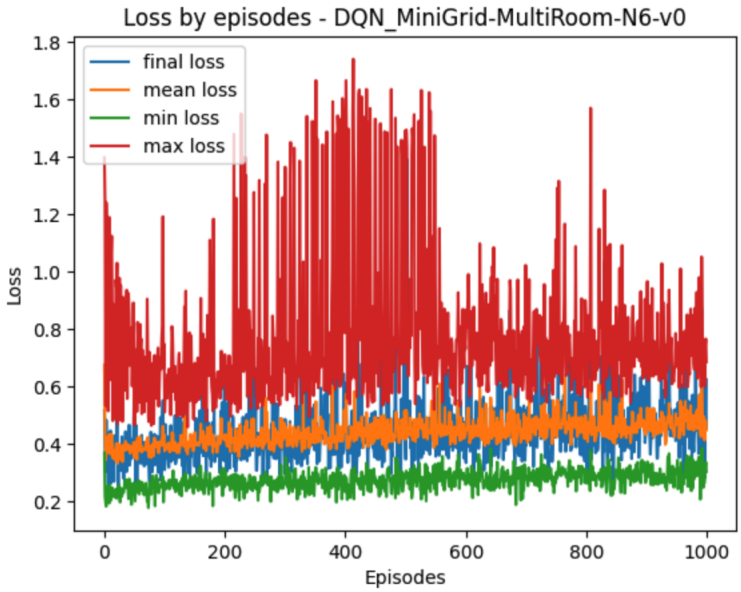
7.5 DQN Large Room - Reward Shaping 2

7.5.1 Figure 10 - Wins

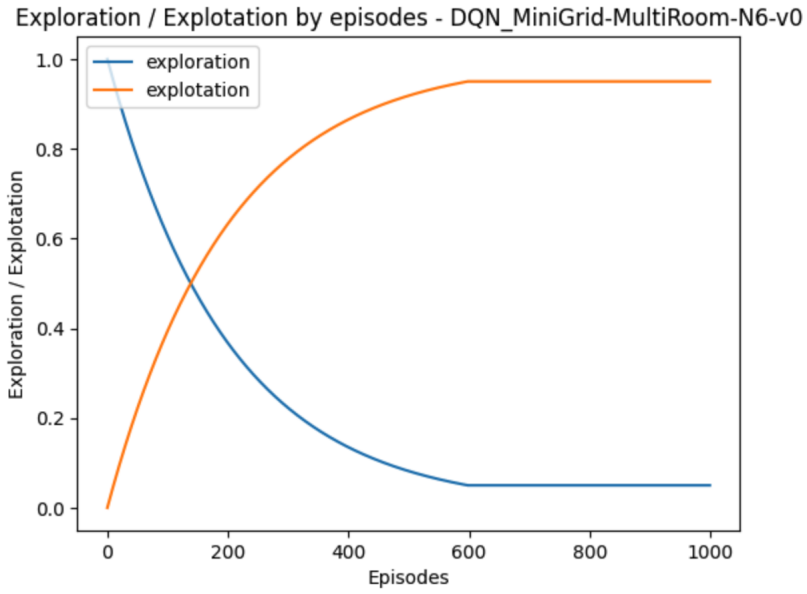


7.5.2 Figure 11 - Reward & Loss



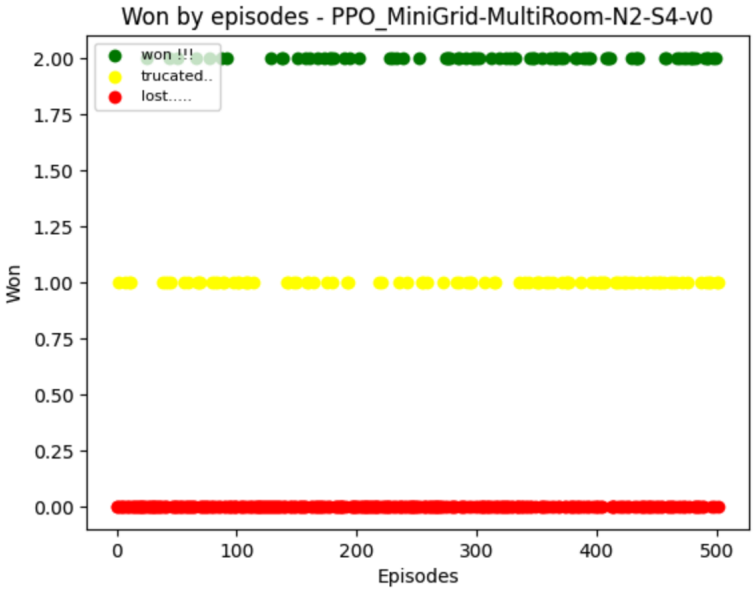


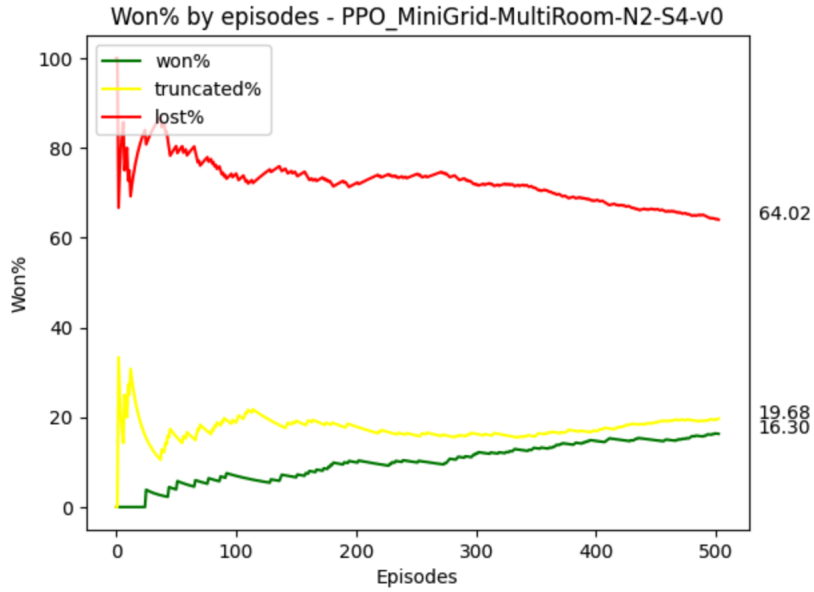
7.5.3 Figure 12 - Exploration / Exploitation



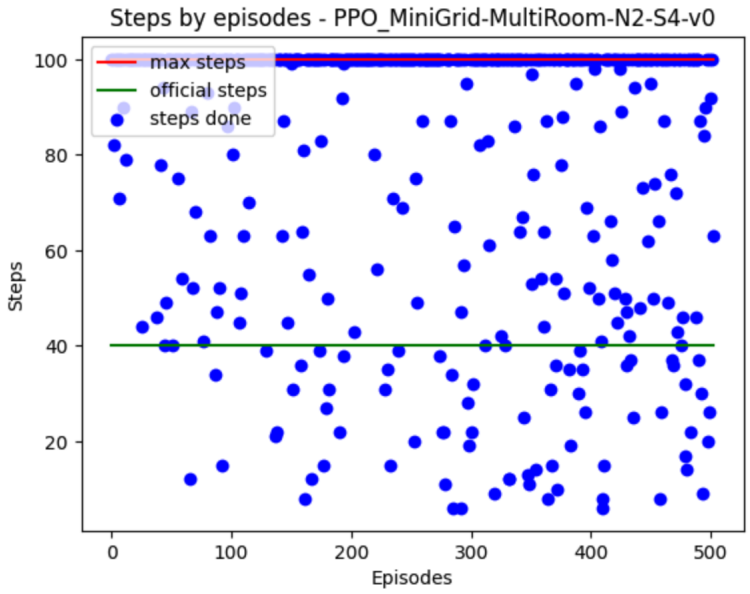
7.6 PPO Small Room - Reward Shaping 2

7.6.1 Figure 13 - Wins

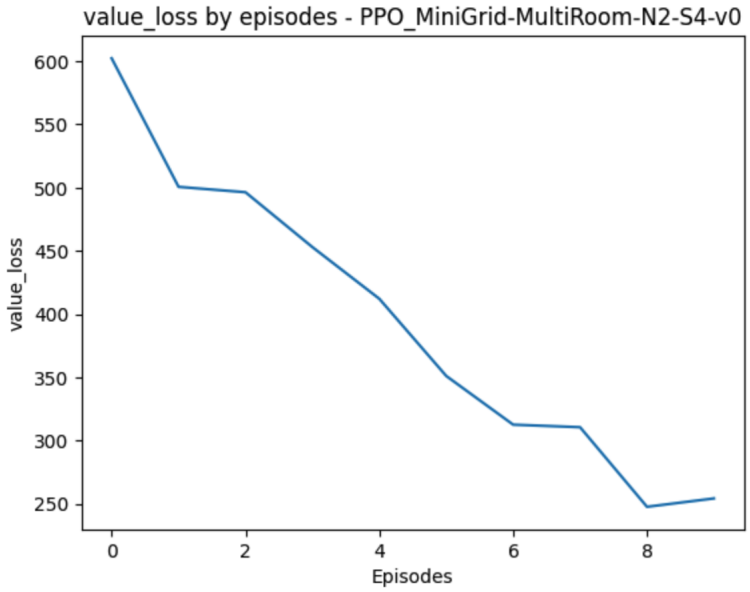




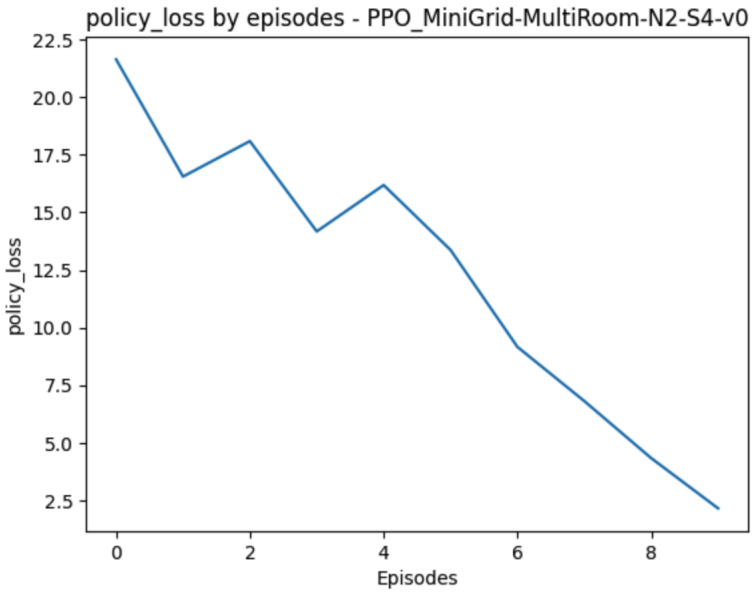
7.6.2 Figure 14 - Steps



7.6.3 Figure 15 - Value loss

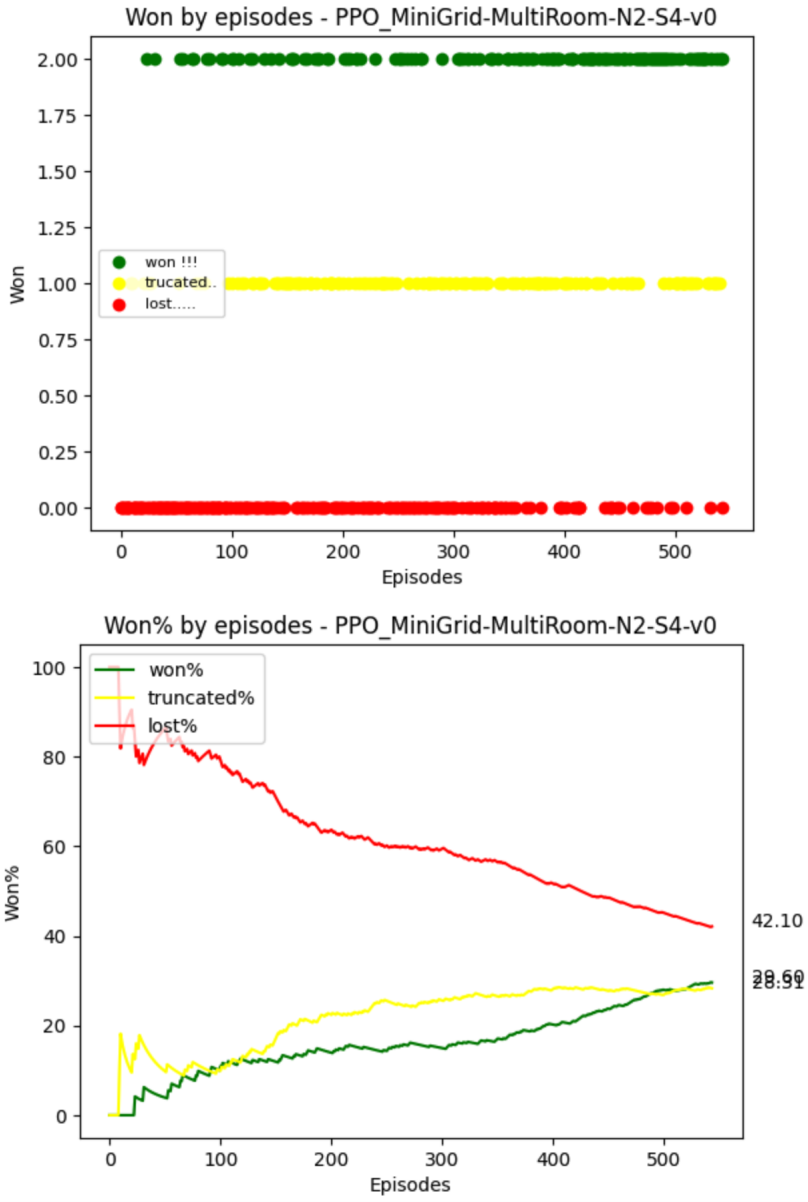


7.6.4 Figure 16 - Policy loss

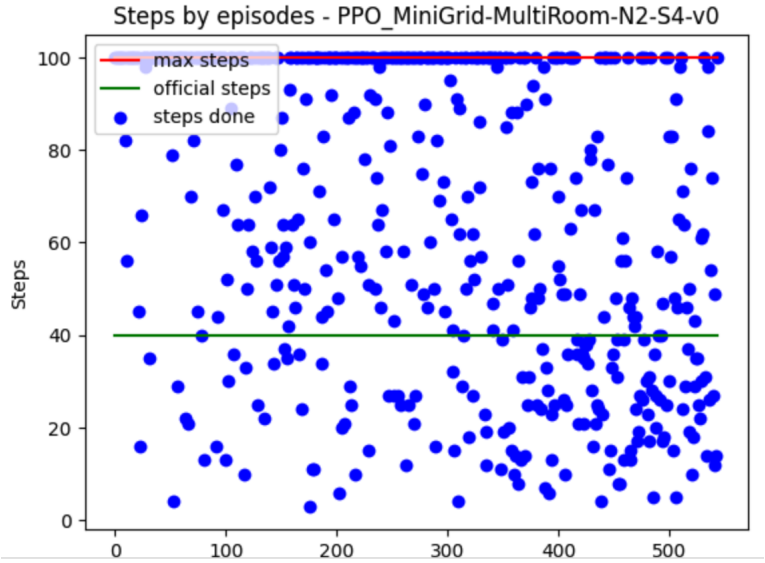


7.7 PPO Small Room - Reward Shaping 5

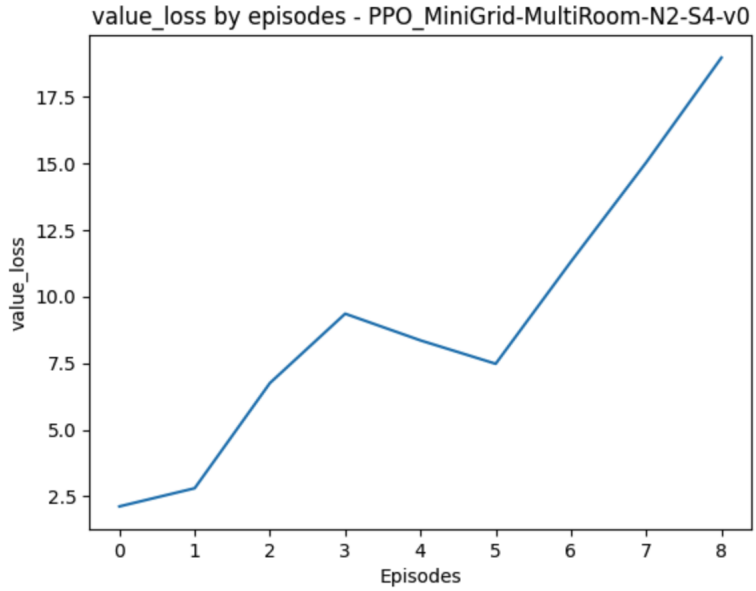
7.7.1 Figure 17 - Wins



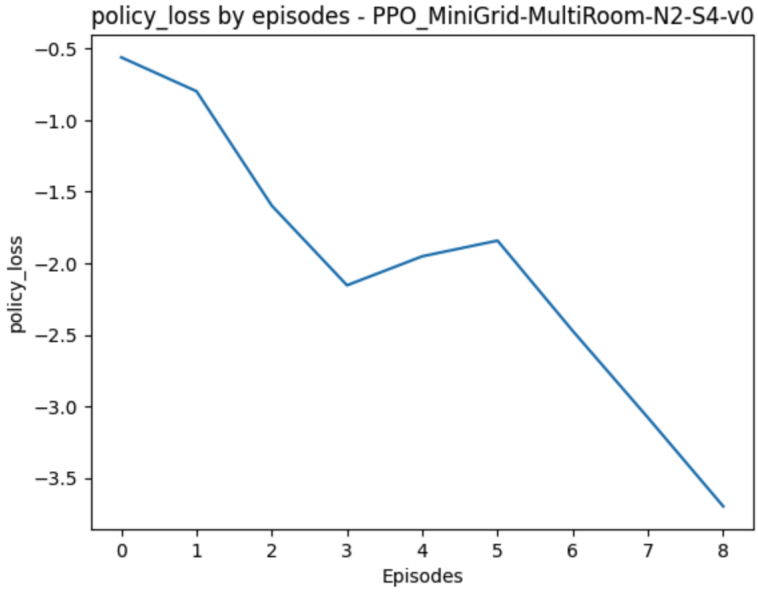
7.7.2 Figure 18 - Steps



7.7.3 Figure 19 - Value loss

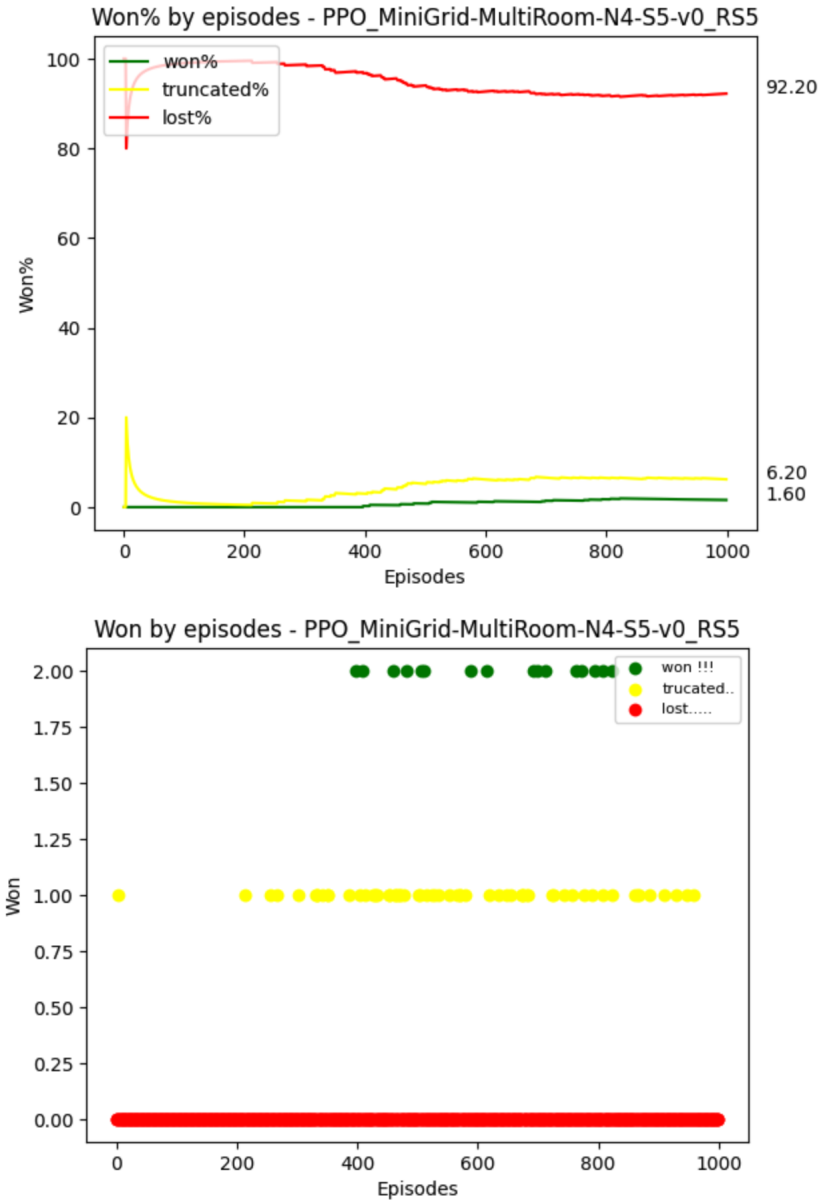


7.7.4 Figure 20 - Policy loss

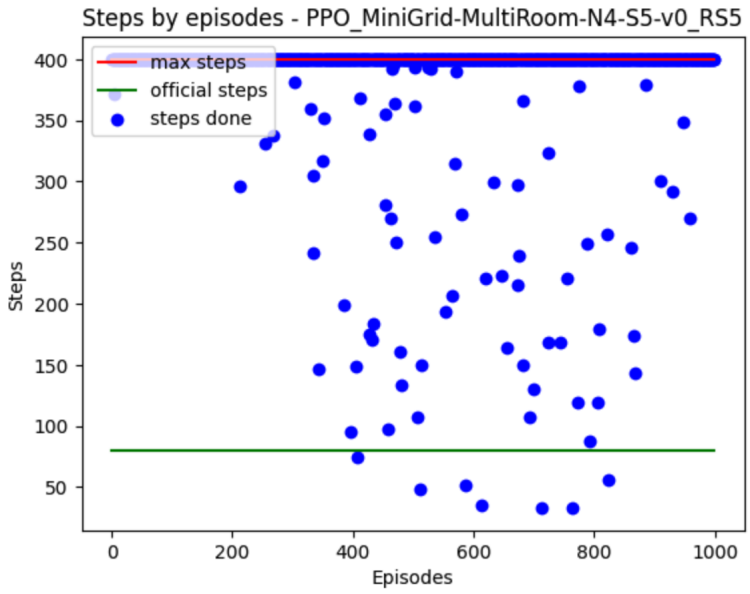


7.8 PPO Medium Room - Reward Shaping 5

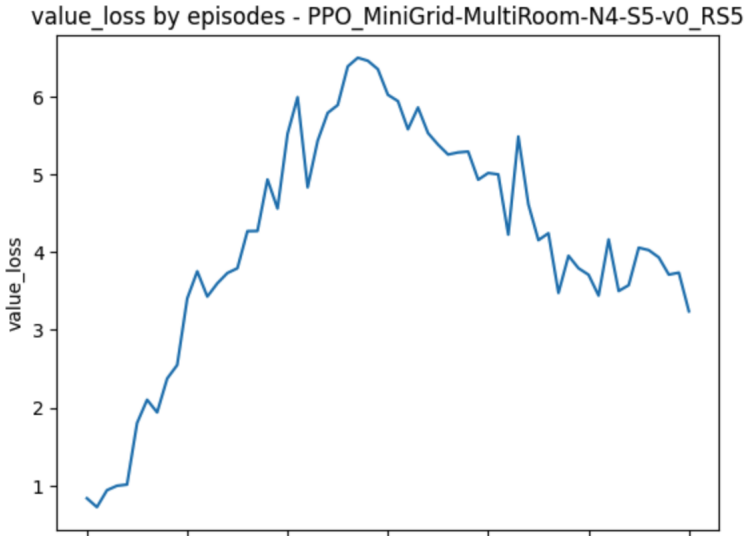
7.8.1 Figure 21 - Wins



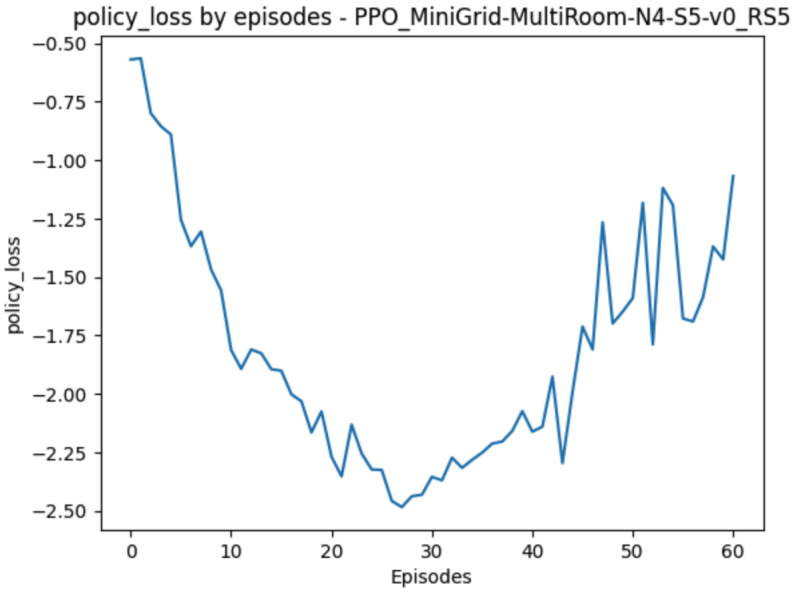
7.8.2 Figure 22 - Steps



7.8.3 Figure 23 - Value loss

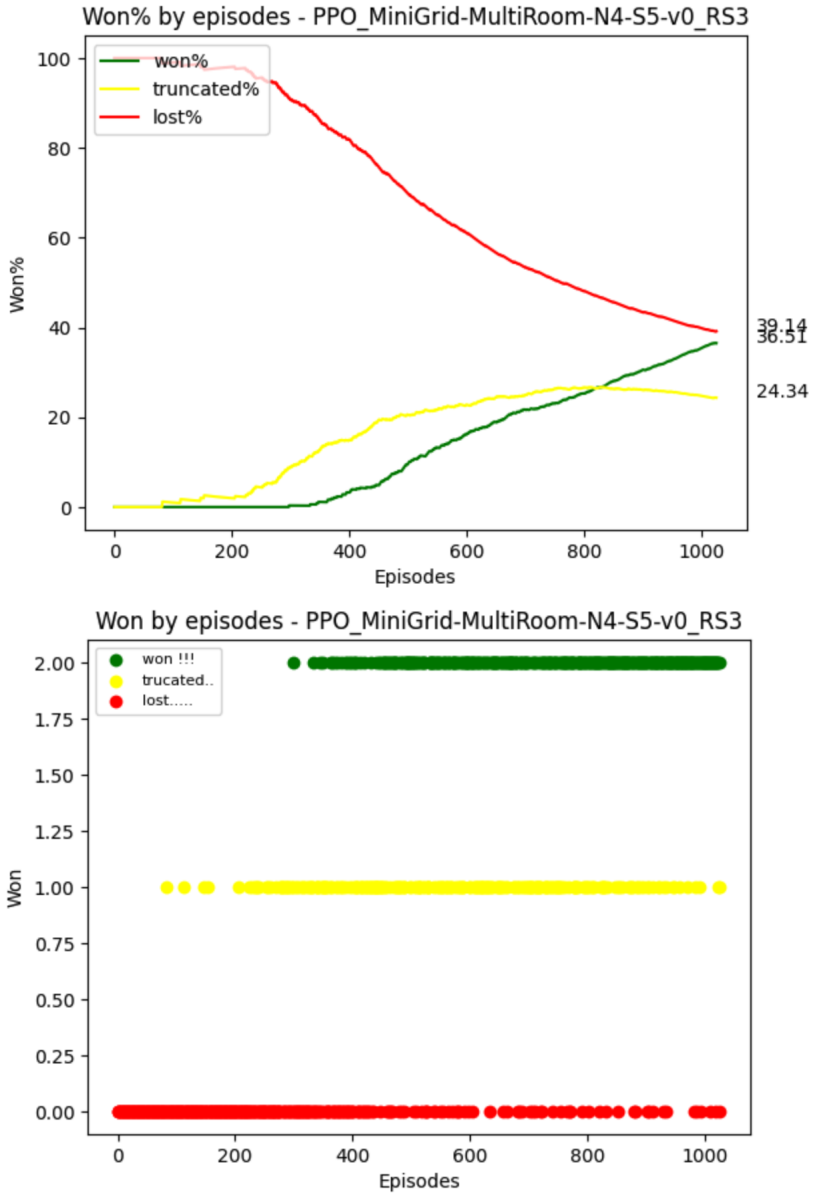


7.8.4 Figure 24 - Policy loss

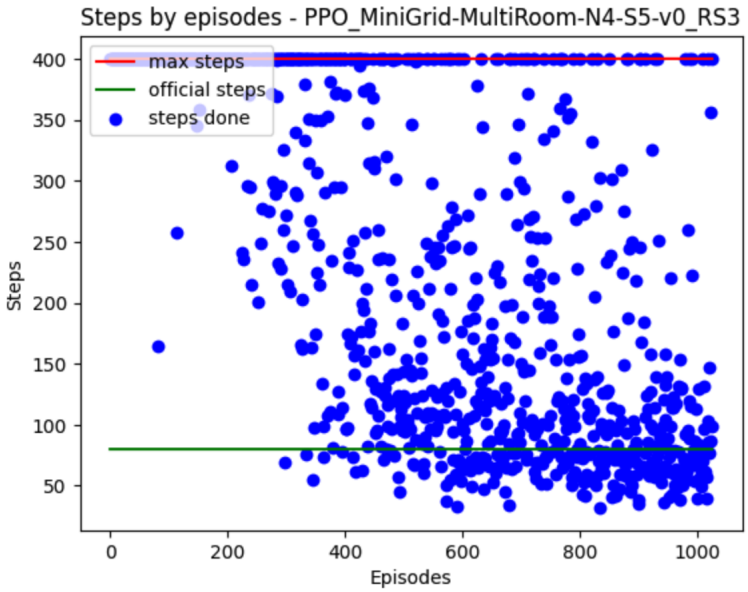


7.9 PPO Medium Room - Reward Shaping 3

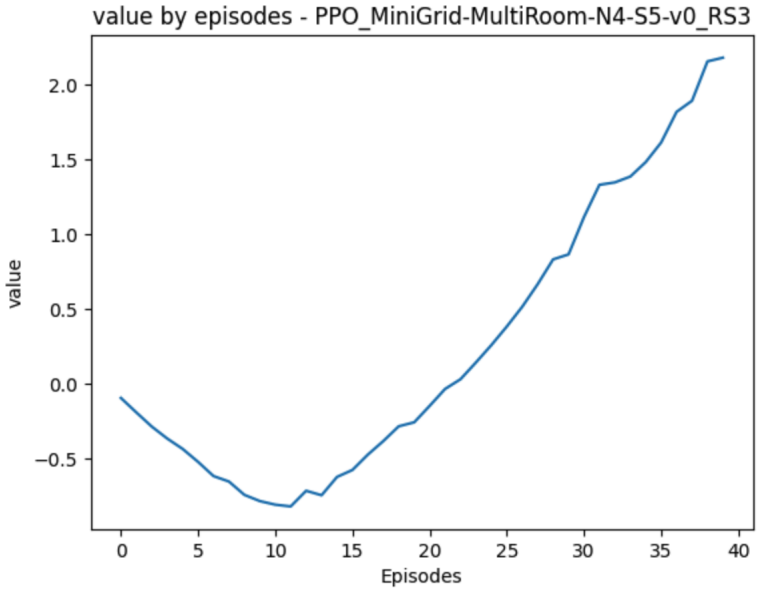
7.9.1 Figure 25 - Wins



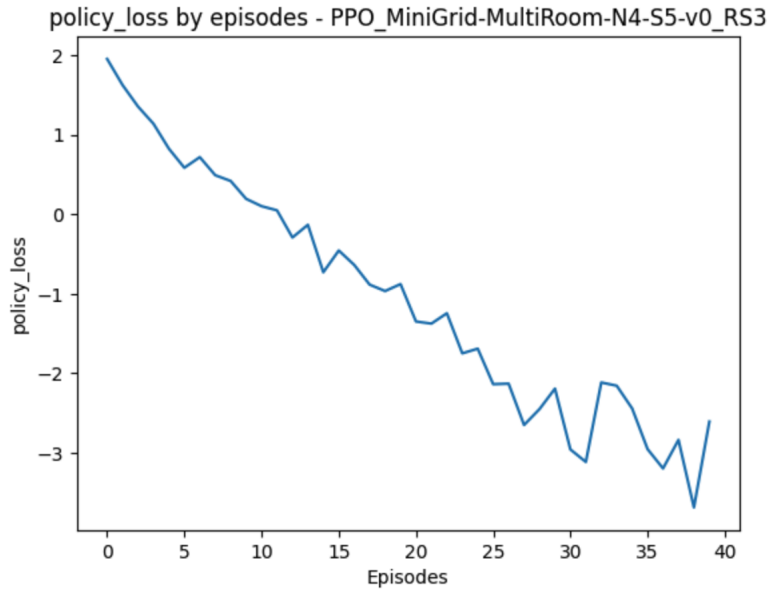
7.9.2 Figure 26 - Steps



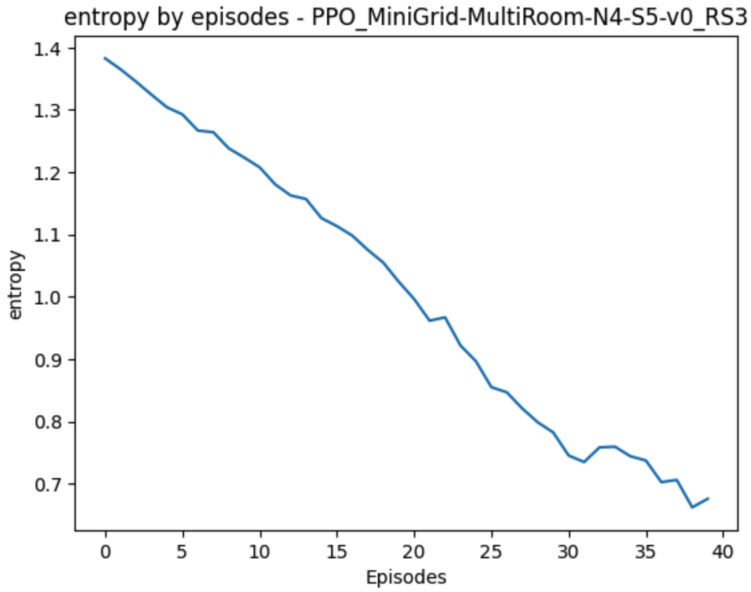
7.9.3 Figure 27 - Value



7.9.4 Figure 28 - Policy loss

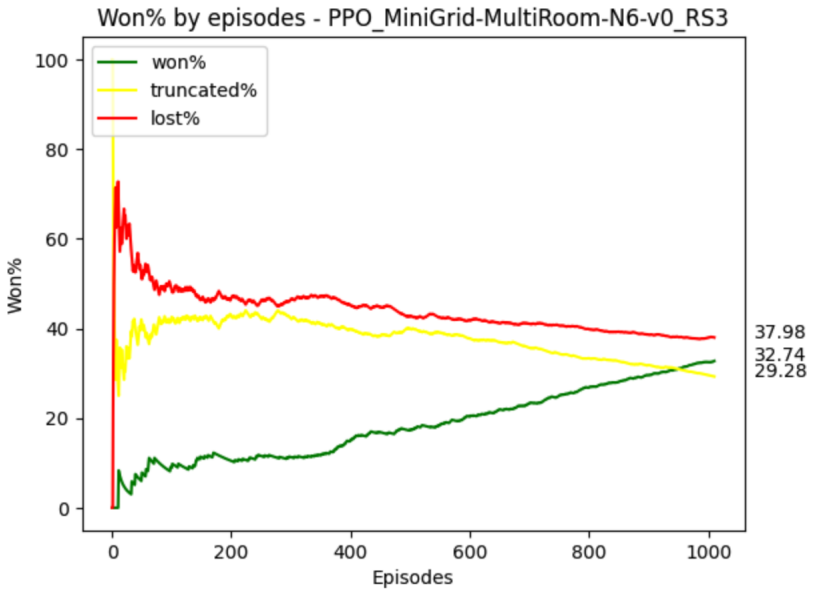


7.9.5 Figure 29 - Entropy

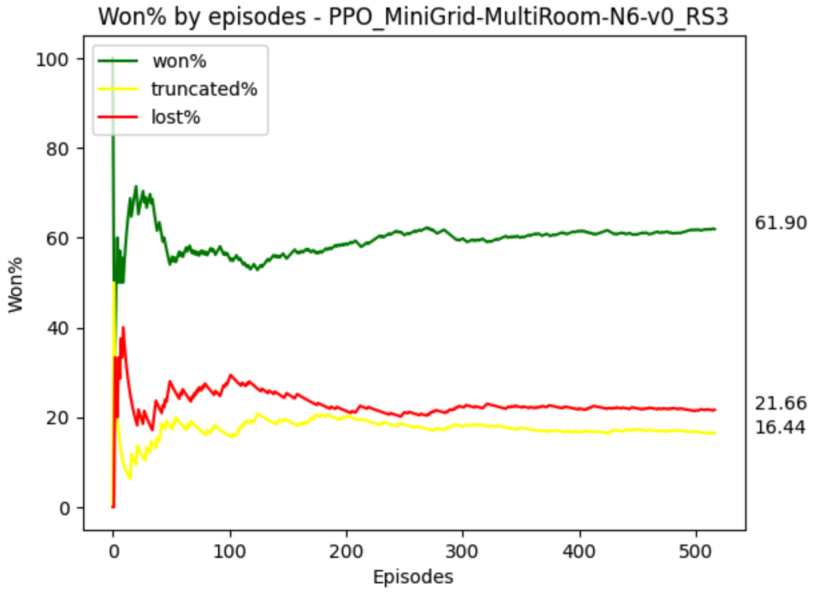


7.10 PPO Large Room - Reward Shaping 3

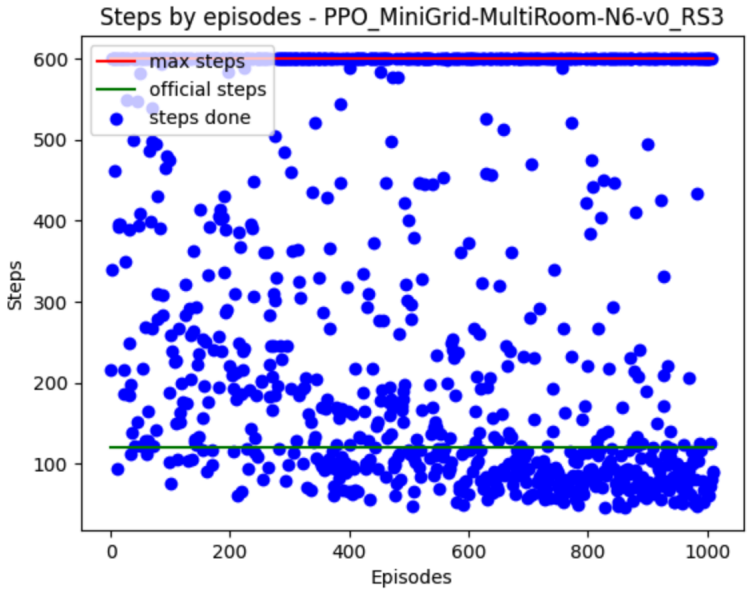
7.10.1 Figure 30 - Wins - 1st stage



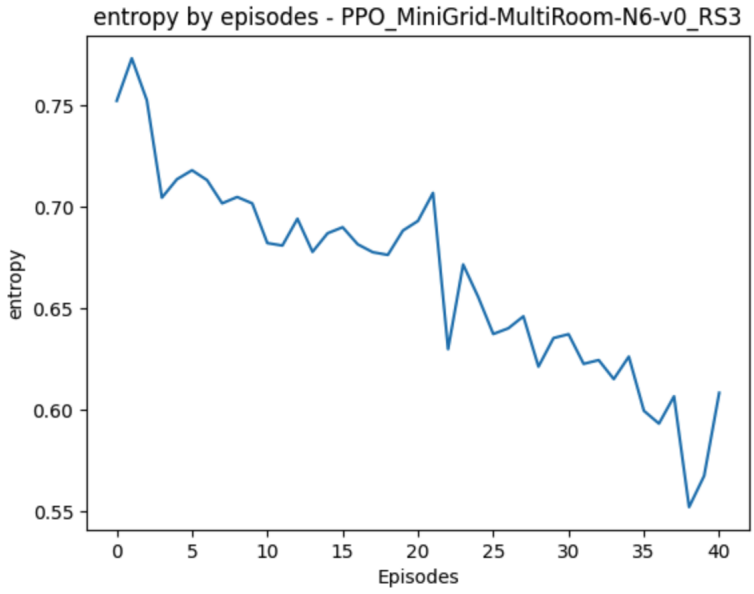
7.10.2 Figure 31 - Wins - 2nd stage



7.10.3 Figure 32 - Steps

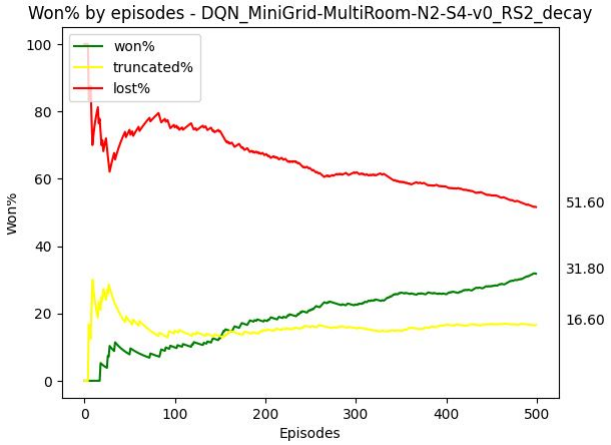


7.10.4 Figure 33 - Entropy

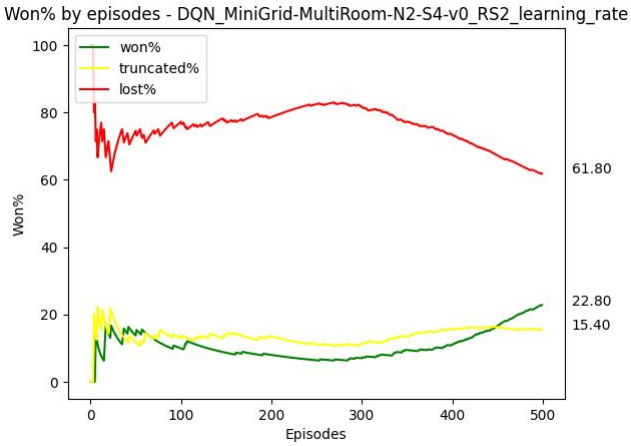


7.11 DQN Hyperparameter Tuning

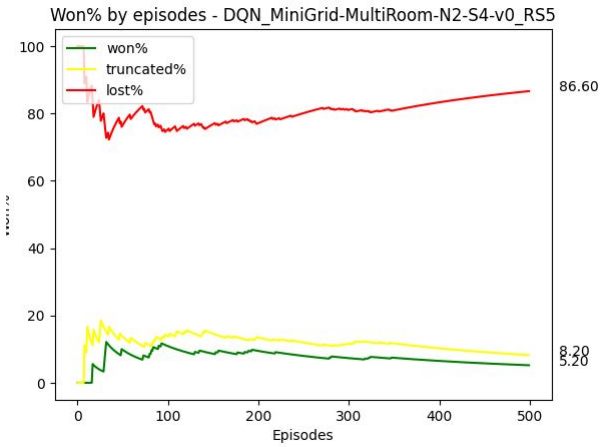
7.11.1 Figure 34 - Lower Decay



7.11.2 Figure 35 - Lower Learning Rate

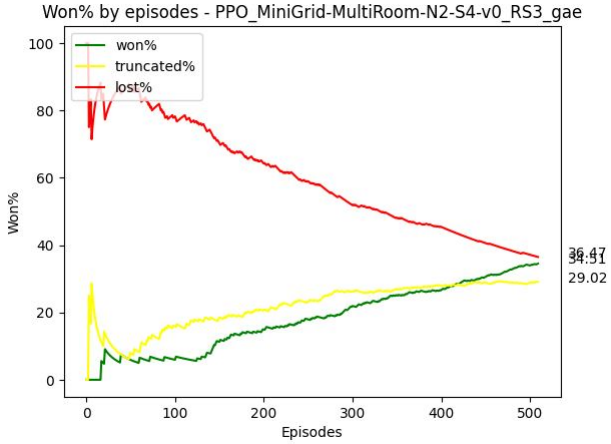


7.11.3 Figure 36 - Target Network

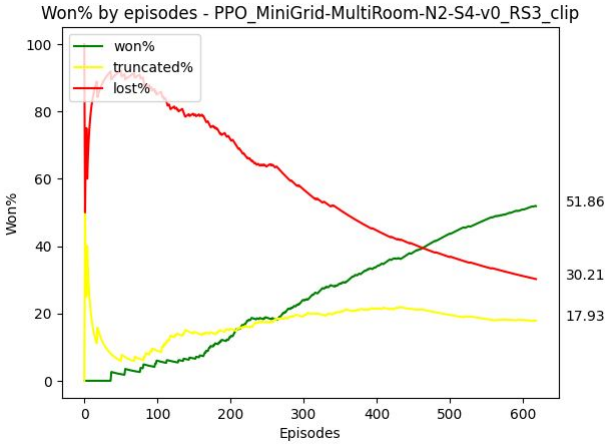


7.12 PPO Hyperparameter Tuning

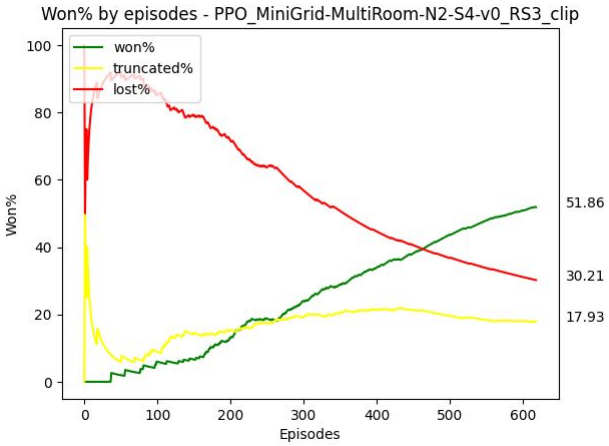
7.12.1 Figure 37 - General Advantage



7.12.2 Figure 38 - Clipping

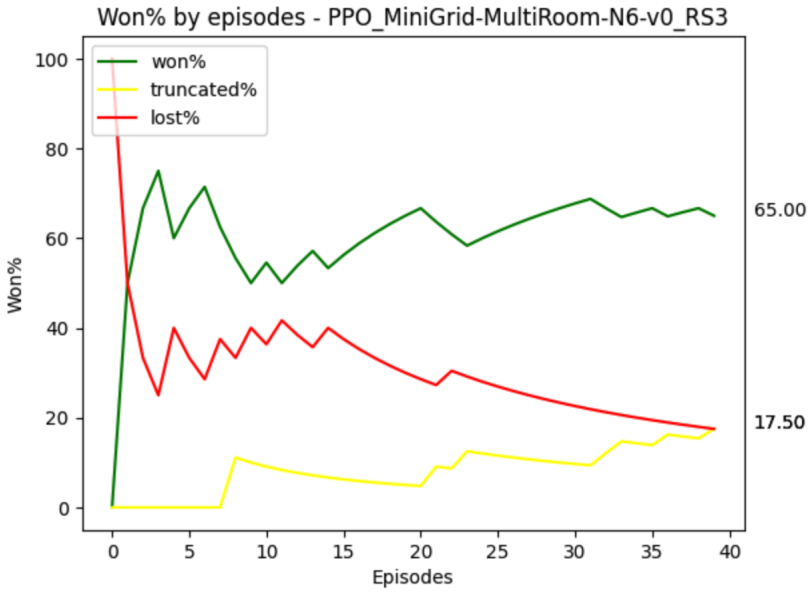


7.12.3 Figure 39 - PPO Epochs



7.13 PPO Large Room - Evaluation

7.13.1 Figure 40 - Wins



7.13.2 Figure 41 - Steps