DDQN and PPO in Super Mario Bros

Daniel Skymoen *daniesky@stud.ntnu.no*
Edvard Schøyen *ecschoye@stud.ntnu.no*
Jarand Romestrand *jarandjr@stud.ntnu.no*
Kristian Vaula Jensen *kristvje@stud.ntnu.no*

Norwegian University of Science and Technology,
Department of Computer Science,
NO-7491 Trondheim, Norway

Abstract

This article emphasizes comparing the sample efficiency, stability and completion rate of DDQN and PPO in a complex environment.

The approach to the project involved implementing the algorithms and training them on OpenAI's (OpenAI) gym-super-mario-bros (Kauten, Christian, 2022) environment. Through repeated experimentation and logging of each of the algorithm's results, the differences in performance between the two have become clear. Both show certain strengths, with DDQN showing reliability with a steady exploration/reward trade-off and lower variance. PPO on the other hand, displays better completion rate, achieving higher rewards at a faster rate. The findings contribute valuable insight into the algorithms' applicability and learning behavior.

The highlight of the project showcases the importance of algorithm selection for reinforcement learning. The source code to reproduce the results is available[1].

[1]The source code to reproduce the results https://github.com/ecschoye/idatt2502-project

1 Introduction

Video games and emulators that allow full control over a high-dimensional sensory input environment are a staple in the research and development of machine learning algorithms. With the field of machine learning rapidly advancing, there remains considerable educational value in exploring and providing fresh perspectives on established algorithms such as DDQN and PPO.

This report focuses on conducting a comparative analysis of DDQN and PPO within the domain of gym-super-mario-bros. The choice of algorithms is motivated by their distinct approaches to learning, track-record in the field, and their similar approaches to neural networks. The comparison focuses on sample efficiency, stability and completion rate. The clear distinctions between the modes of operation makes comparative analysis challenging. In consideration of this challenge, the report also focuses on defining the context and terms for the analysis.

2 Terminology

ANN. *Artificial neural network consists of a set of layers with nodes/artificial neurons and weights that connect them to each other. ANN aims to mimic the information processing of the brain.*

DNN. *Deep Neural Network is a type of ANN that consists of many hidden layers. This enables it to learn complex hierarchical representations from data.*

DQN. *Deep Q-Network consists of one DNN for estimating the Q-value depending on state and a given action.*

DDQN. *Double Deep Q-Network is similar to DQN but instead of one DNN it consists of two networks. The policy network (Q-network) used to select the best action at each step, and the target network used to estimate the Q-value of that selected action.*

Policy. *The policy defines the probability distribution over actions given a certain state.*

Softmax. *Softmax is an activation function often used in ANN to transform raw output of a network to a probability distribution with values from 0 to 1.*

ReLU. *Rectifier linear unit is an activation function often used in ANN's hidden layers to introduce non-linearity . This function outputs the input value if it is positive and zero otherwise.*

PPO. *Proximal Policy Optimization is a policy optimization algorithm that updates the policies conservatively to converge to an optimal solution. PPO iteratively updates its policies based on batches of experiences collected from interactions with the environment.*

Sample efficiency. *Sample efficiency describes the amount of experience an agent needs to generate in an environment, in order to achieve a certain level of performance.*

3 Related Work

In the field of deep reinforcement learning in hardcore environments, extensive research has been conducted that provides valuable information.

A significant contribution in the field of DDQN was the first use of deep q-learning on a hardcore environment performed by Deepmind researchers (Mnih et al., 2013). Another relevant research is the introduction of DDQN as a solution to the overestimation issues of DQN (van Hasselt et al., 2015).

The most significant work related to PPO is the actual development of the set of PPO algorithms done by researchers at OpenAI. They released the paper describing this new set of policy oriented algorithms, showcasing the algorithm's strengths on simple and complex environments. (Schulman et al., 2017). Following this, OpenAI has demonstrated the uses of PPO on other high-dimensional environments such as Montezuma's Revenge (Salimans & Chen, 2018).

4 Methods

4.1 Environment

Emulation of the environment was done using gym-super-mario-bros, a code library built on OpenAI's Gym. Within this environment, objectives and rewards are already managed, and diverse sets of actions are allowed. Various preprocessing techniques were used to optimize the data frames for training. This included actions such as downscaling, greyscaling, and frame skipping. To limit the experimental factors, the simplified action set was chosen, and the scope was narrowed down to two distinct stages within the environment.

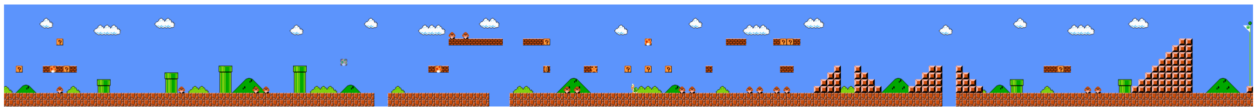


Figure 1: World 1-1. Source: The Nintendo Maps

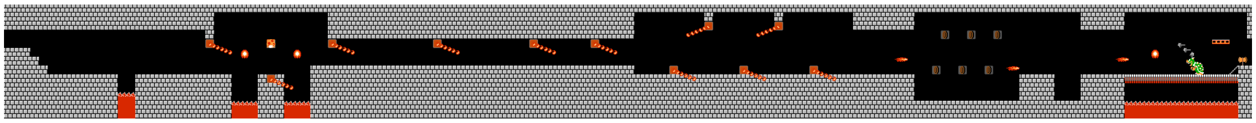


Figure 2: World 6-4. Source: The Nintendo Maps

4.2 Network Architecture

Both algorithms implement the same neural network architecture. It consists of a sequence of convolution layers using ReLU activation to process the frames provided by the environment. The output of the convolutional layers is then flattened and passed through a sequence of fully connected layers. The final layer has an output size usually corresponding to the number of discrete actions.

4.3 DDQN

In this project DDQN was roughly implemented as detailed in "Playing Super Mario Bros with Deep Reinforcement Learning" (Chao, 2021).

4.3.1 Exploration Strategy

In DDQN, an epsilon-greedy policy is used to balance exploration and exploitation. It attempts to keep the algorithm from prematurely converging to sub-optimal strategies. The exploration rate is calculated using an exponential decay function. The factors which determine the range and speed of the decay is as follows:

- ep_{min} - The lower end of the epsilon range.
- ep_{max} - The upper end of the epsilon range.
- decay - A constant which determines the speed of decay from max to min. For the runs conducted in this project, the range lies between 10^5 to 10^6, depending on the number of episodes.
- steps - The total count of steps for the training run so far.

$$epsilon = ep_{min} + (ep_{max} - ep_{min}) \times e^{-\frac{steps+1}{decay}}$$

4.4 PPO

"Code PPO from scratch with PyTorch" (Yu, 2020) explains the process of implementing the fundamental elements of the PPO algorithm. This implementation was based on continuous state- and action spaces. Therefore, changes to the way actions are fetched and evaluated was made to fit the discrete action space of the environment. To calculate actions, the algorithm fetches probabilities of each action from the actor network. Afterwards, an action is sampled from a distribution of these probabilities with Softmax activation. To safeguard against exploding gradients, the algorithm only allows updating the policy if the approximation of the updated policy is below a parameter *target_kl*.

4.5 Logging

Central to the development of the machine learning models was the use of Neptune AI (Neptune Labs), a machine learning tool designed for logging, organizing and presenting machine learning data. During the early stages of development, the logger was used to troubleshoot by monitoring the data. In later stages, the logs were used to fine tune parameters, allowing comparison between different runs.

4.6 Context of analysis

It was crucial to find the right context and training configuration for the experiments to ensure a balanced comparison between the two algorithms. This entailed defining what metrics should be used in training, and figuring out how to present the results in a comparative nature without the differences in the algorithms having a negative impact.

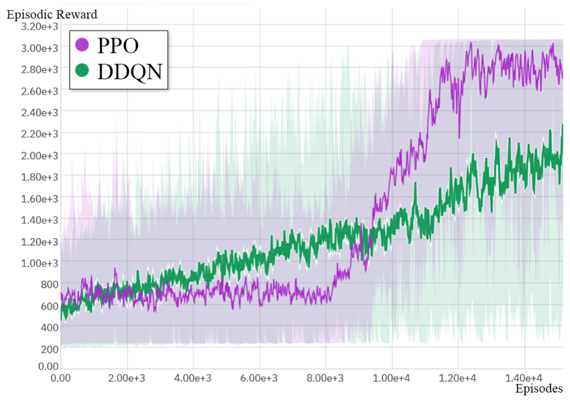
The analysis is based on number of episodes. This choice was logical, due to the focus on sample efficiency. An episode in this context, consists of the agent having three lives to attempt to get the flag, which is obtained when a stage is completed. If the agent obtains the flag, the episode ends.

Due to the limited time and resources, the experiments has a frame of 15 000 episodes. This number of episodes allowed us to find good context for comparison in terms of results, sample efficiency and stability, without challenging the time frame.

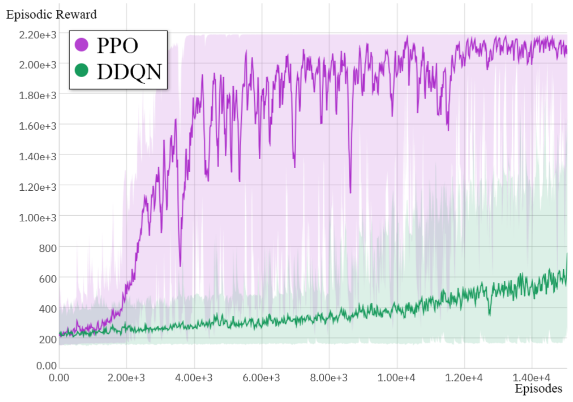
5 Results

The results are presented using data collected through Neptune logging. Comparing the two algorithms will mainly focus on episodic reward and completion rate analytic.

5.1 Episodic Rewards



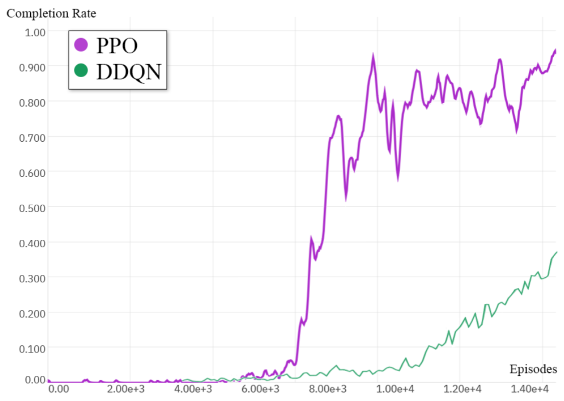
(a) **Episodic Reward, Stage 1-1.**



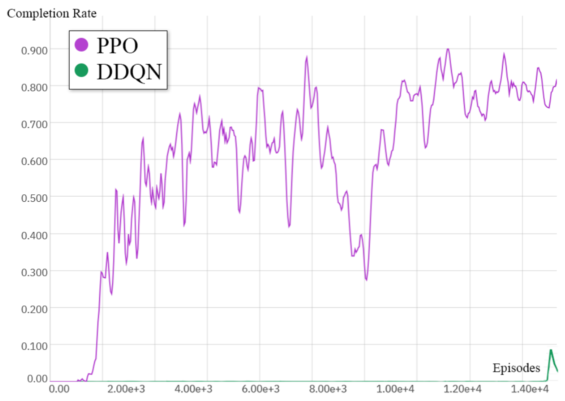
(b) **Episodic Reward, Stage 6-4.**

Figure 3: Graphs displaying rewards per episode for both algorithms on the two selected stages. Training in such a complex environment often leads to a large variance in rewards, especially at the start of the training process. Therefore, the graphs are smoothed to better display the algorithms' development throughout the training process.

5.2 Completion Rate



(a) **Completion Rate, Stage 1-1.**



(b) **Completion Rate, Stage 6-4.**

Figure 4: Comparative graphs showing completion rate on both maps. Completion Rate is the rate at which the actor is able to complete the course. Each algorithm keeps track of the last 100 episodes and logs this throughout the training process. The graphs are smoothed.

6 Discussion

When comparing DDQN and PPO, their different mode of operation must be taken into consideration. Looking at their performance with regard to sample efficiency shows the algorithms' ability to learn from limited data. Other points up for discussion are completion rate and consistency within the environment. Utilizing this analysis allows for evaluation and comparison of these algorithms.

6.1 Completion rate

To discuss the completion rate, attention is directed towards the analytic *flag_average*, which is defined as the average number of flags achieved/stages completed during the last 100 episodes.

Stage 1-1 showcases significant measurable results for both algorithms. In terms of completion rate PPO achieved a rate exceeding 90% and looks to converge towards ideal results given enough training. In contrast, DDQN achieved a rate ranging from 40% and 60%, exhibiting some variance. The data seems to indicate that DDQN has a consistent growth. However, more troublesome is the variance in completion rate, and required number of episodes. This makes achieving perfection with DDQN less likely.

In stage 6-4, PPO shows a completion rate rising toward 80%, indicating a probable ability to master the stage, given sufficient training. On a different note, DDQN struggles at this stage, achieving only a maximum of 0.05 % completion rate. The reward graph also showcases poor results. This could be explained by the complexity of the stage or by a number of factors in implementation and hyperparameter tuning. There are some solutions that might improve DDQN's performance on the stage. A possible improvement is modifying the epsilon function to raise the epsilon if the agent does not achieve higher reward over a period of time. This could enhance its ability to explore the later section of the stage.

6.2 Sample efficiency

In terms of sample efficiency there are noticeable differences between the two algorithms, but also a disparity between the stages. Looking at the episodic reward graph of stage 1-1, DDQN starts achieving higher rewards earlier as it progresses somewhat linearly. PPO on the other hand, catches up and reaches a higher reward ceiling as the reward graph looks more like the activation function softmax. Once PPO starts learning the environment it looks a lot more sample efficient than DDQN.

In stage 6-4, PPO achieves results earlier, while DDQN progresses slowly. The results from stage 1-1 may lay a better foundation for comparison due to the poor results of DDQN on 6-4. One could argue that DDQN has better sample efficiency, due to the more proportional relationship between experience and reward. On the other hand, PPO receives higher results in the total training frame.

6.3 Stability

In the context of the analysis, stability denotes how stable the growth of rewards and flag averages is. Here, the stability can be determined by studying the graphs, looking for a consistent ascending trend in rewards, and less regressive patterns indicating potential setbacks in the training process. With this in mind the graphs show a significant difference in the algorithms' stability. The PPO shows less stability for both stages due to its quick progress, exploring a variety of policies in the early stages of convergence. This behavior can be altered by tuning the hyperparameters. However, variance is not a concern as long as it normalizes towards the optimal policy at the later stages of training.

For DDQN there is a noticeable increase in stability. This makes sense as DDQN's double neural network solution is designed to stabilize the algorithm. For DDQN the data displays more variance later on in the data set, making sense because its utilizing more of the agents experience but still performing some random movements causing it to have a wider spread in rewards. There is some variance occurring in the completion rate. This could be explained by remaining exploration, or because it still attempts to learn even when focusing on exploitation. This could cause some dissonance in what actions the agent performs, and produce the variance observed.

7 Conclusion

Both algorithms have proven to be good alternatives in hardcore environments. The results indicates that PPO seems like the better choice as it achieves significantly better results in terms of completion rate and reward. DDQN does demonstrate some strength in sample efficiency due to its very consistent reward/experience relationship, but PPO achieves generally higher rewards in the total sample size. The data indicate that DDQN is the more stable algorithm, but PPO trades slight instability for much higher performance. Therefore, which algorithm is the best choice depends on what factors one values. If more stable learning with a constant trade-off between experience and reward is wanted, DDQN could be the best option. If one focuses on the actual reward, and desires fast results then PPO is the optimal algorithm.

8 Future Work

Future work may involve exploring some of the ideas presented in discussion such as further development of the epsilon function. Due to the big difference in DDQN results on the two stages, complex stages could be subject for further examination. Another interesting idea is comparing how PPO and DDQN perform in environments with much larger or concurrent action spaces. Utilizing methods to find optimal hyperparameters, such as Grid Search is an interesting possibility. These focused investigations can contribute valuable insights to the field of reinforcement learning.

References

De-Yo Chao. Playing super mario bros with deep reinforcement learning. *Analytics Vidhya*, 2021. URL `https://www.analyticsvidhya.com/blog/2021/06/playing-super-mario-bros-with-deep-reinforcement-learning/`.

Kauten, Christian. gym-super-mario-bros. `https://pypi.org/project/gym-super-mario-bros/`, Accessed 2023.27.10, 2022.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *Google DeepMind*, 2013.

Neptune Labs. Neptune ai. `https://neptune.ai/`, Accessed 2023.27.10.

OpenAI. Gymnasium. `https://gymnasium.farama.org/`, Accessed 2023.27.10.

Tim Salimans and Richard Chen. Learning montezuma's revenge from a single demonstration. *OpenAI*, 2018.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *OpenAI*, 2017.

The Nintendo Maps. The nintendo maps. `https://nesmaps.com/maps/SuperMarioBrothers/SuperMarioBrothers.html`, Accessed 2023.23.11.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *Google DeepMind*, 2015.

Eric Yang Yu. Coding ppo from scratch with pytorch. *Medium*, 2020. URL `https://medium.com/analytics-vidhya/coding-ppo-from-scratch-with-pytorch-part-1-4-613dfc1b14c8`.

A Appendix 1: Other Findings

Some additional findings and results beyond the scope of rewards and completion rate.

A.1 Loops

Both DDQN and PPO are capable of learning and completing stages within the gym-super-mario-bros (Kauten, Christian, 2022) environment. However, an interesting observation was made when training the models on the final stage of the game. As shown in the image below, the agent must enter one of the tubes to progress. Should he instead continue forward, the agent will be stuck in a loop, constantly being teleported back to start. '*B*' marks the spot where the actor is teleported back to start: Since change in position is

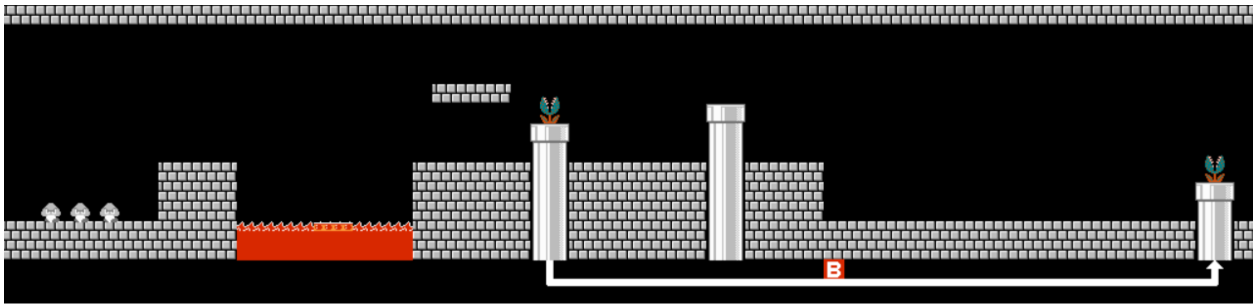


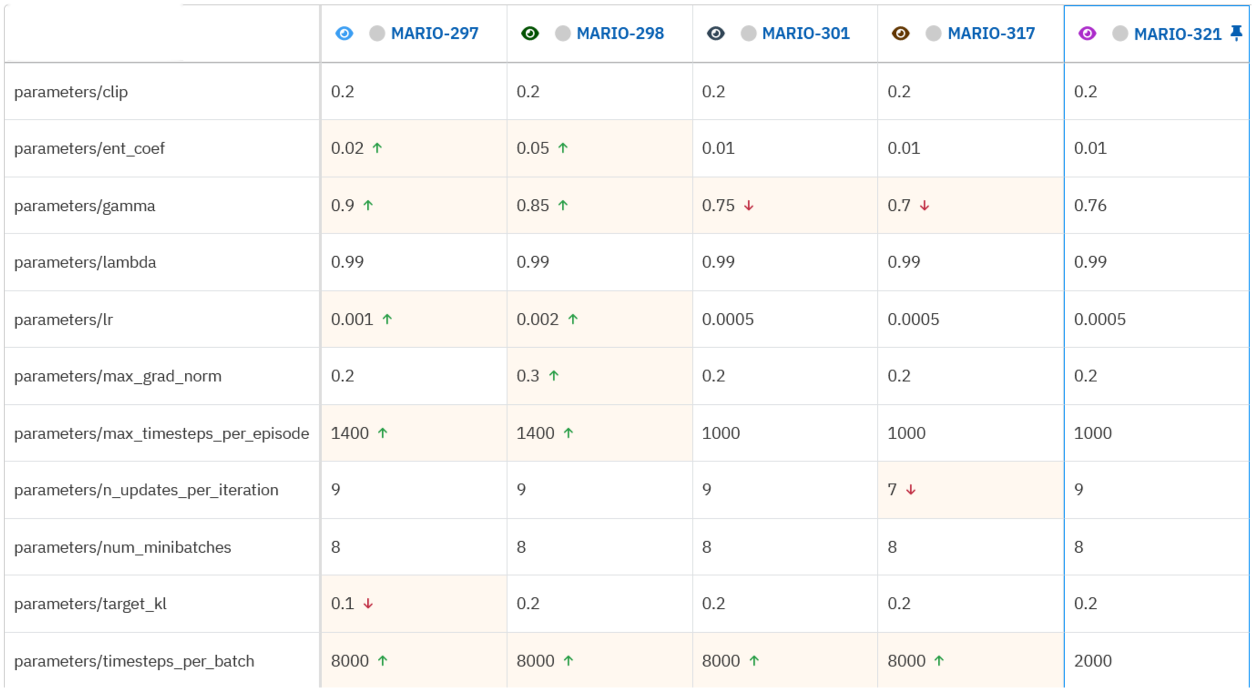
Figure 5: World 8-4 Loop. Source: The Nintendo Maps

one of the factors providing rewards, the algorithms quickly learn to pace past the obstacles and therefore gets stuck in this loop. Whenever the agent hits the first Piranha Plant and dies, it tries to avoid future engagements with the obstacle. This makes it very unlikely that future actions will lead to it exploring the correct way to progress through the episode. Both PPO and DDQN agents got stuck in this infinite loop.

A.2 Significance of Hyperparameter Tuning

One of the biggest challenges of a machine learning project like this, is tuning the different parameters to give us the optimal outcome. Getting enough training is time consuming, and even the slightest change to certain parameters can have huge impact on results.

The substantial lowering of *timesteps_per_batch* turned out to have a very positive impact on the training process on this particular stage. In the PPO implementation, this parameter decides how many timesteps of data collected in each rollout. Normally the algorithm collected as much data as possible in regards to available memory on the hardware that got trained on. However, for this particular run, *MARIO-321*, the smaller batches made the model able to progress and even master the stage towards the end of the training session. There might be different reasons to why this had such a great impact on the training. Reducing the rollout data collection might have helped the model focus more on the early dynamics of the stage, letting it master the stage step by step instead of over-fitting to later dynamics.



	MARIO-297	MARIO-298	MARIO-301	MARIO-317	MARIO-321
parameters/clip	0.2	0.2	0.2	0.2	0.2
parameters/ent_coef	0.02 ↑	0.05 ↑	0.01	0.01	0.01
parameters/gamma	0.9 ↑	0.85 ↑	0.75 ↓	0.7 ↓	0.76
parameters/lambda	0.99	0.99	0.99	0.99	0.99
parameters/lr	0.001 ↑	0.002 ↑	0.0005	0.0005	0.0005
parameters/max_grad_norm	0.2	0.3 ↑	0.2	0.2	0.2
parameters/max_timesteps_per_episode	1400 ↑	1400 ↑	1000	1000	1000
parameters/n_updates_per_iteration	9	9	9	7 ↓	9
parameters/num_minibatches	8	8	8	8	8
parameters/target_kl	0.1 ↓	0.2	0.2	0.2	0.2
parameters/timesteps_per_batch	8000 ↑	8000 ↑	8000 ↑	8000 ↑	2000

Figure 6: **PPO Params, Stage 1-1.** Overview of parameters for different PPO runs on the first stage.

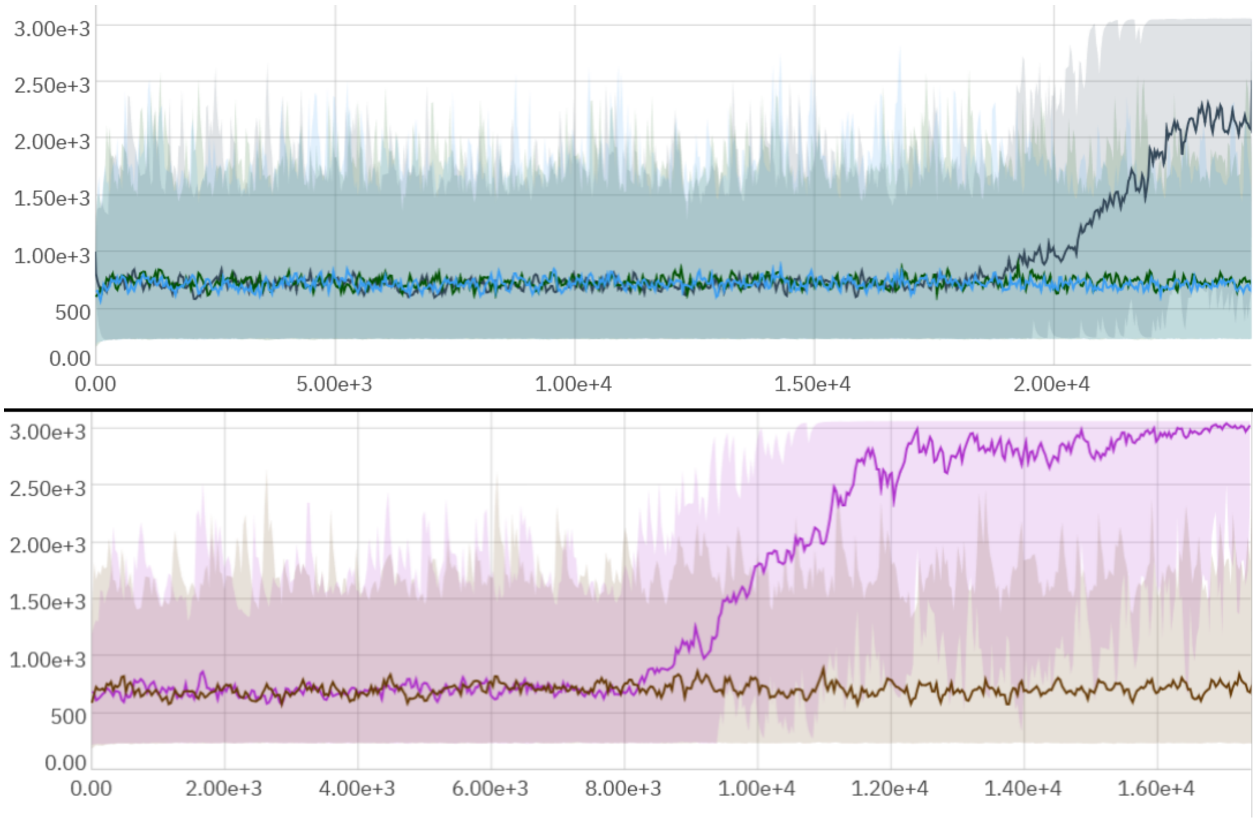


Figure 7: **PPO Rewards, Stage 1-1.** Rewards for each run with corresponding colors to the parameter table above.