Deniz Karakay - 2443307 Date: 11/06/2023

EE449 - Homework 3

Saving the Princess with a Reinforcement Learning Agent

1. Basic Questions

Reinforcement Learning and Supervised Learning are two different approaches to machine learning. In Reinforcement Learning, an agent is the decision-making entity that interacts with the environment, taking actions and receiving feedback through rewards or penalties. The environment is where the agent operates, providing input through rewards or punishments based on the agent's actions. A reward is feedback the environment offers to the agent, indicating the desirability of an action taken. A policy is the agent's strategy or rules to determine efforts to take in a given situation. Exploration is trying out new activities or strategies to discover their potential rewards and improve the agent's understanding of the environment. Exploitation is using the agent's current knowledge to take actions that maximize the expected rewards.

On the other hand, Supervised Learning involves a model or algorithm that learns from labeled data to make guesses. The environment can be considered the dataset or problem domain the model is trying to learn from and predict. The equivalent concept to a reward in Supervised Learning is the loss function, which measures the difference between the model's predictions and the actual labels, guiding the learning process. The nearest concept to a policy in Supervised Learning would be the learned function or mapping from inputs to outputs that the model acquires during training. The equivalent idea to exploration in Supervised Learning is model selection or hyperparameter tuning, where different model architectures or settings are tested to find the best-performing one. Finally, the equivalent concept to exploitation in Supervised Learning is making predictions or inferences using the learned model based on the knowledge acquired during training.

Both Reinforcement Learning and Supervised Learning have their strengths and weaknesses. Reinforcement Learning is practical when the environment is dynamic, and the agent needs to adapt to changing circumstances. It is also useful when the rewards are not known in advance and must be learned through trial and error. Supervised Learning, on the other hand, is valuable when the problem domain is well-defined and labeled data is available. It is also useful when the goal is to make predictions based on the learned model. In summary, both approaches have their place in machine learning, and the choice of which one to use depends on the problem.

2. Benchmarking & Discussions

I utilized Matplotlib to plot my data. I checked them on Tensorboard then export the data as CSV to have nicer plots with legends, title etc.

2.1. Models and Preprocessing Methods

2.1.1. Preprocessing Types

Table 1. Types and details for Preprocessing Methods

Type #	GrayScaleObservation (keep_dim)	VecFrameStack (n_stack)	VecFrameStack (channel order)
Type 0	True	4	"last"
Type 1	False	2	"last"
Type 2	True	2	"first"

2.1.2. PPO Models

Table 2. PPO models and their parameters

Models	Preprocessing Types	Policy	Learning Rate	n_steps	ent_coef	gamma	n_epochs
PPO1	Type 0	CnnPolicy	0.000001	512	0	0.99	10
PPO2	Type 1	MlpPolicy	0.000015	1536	0	0.99	10
PPO3	Type 2	CnnPolicy	0.000005	1024	0.01	0.99	8

2.1.3. DQN Models

Table 3. DQN models and their parameters

Models	Preprocessing Types	Policy	Learning Rate	Batch Size	Exploration Fraction	Exploration Final Eps	Train Freq
DQN1	Type 0	CnnPolicy	5e-3	192	0.1	0.1	8
DQN2	Type 1	MlpPolicy	10e-3	150	0.2	0.1	4
DQN3	Type 2	CnnPolicy	20e-3	165	0.3	0.15	8

2.2. Benchmarking

2.2.1. PPOs Episode Reward Mean

As it can be seen from *Figure 1*, PPO2 has the best Episode Reward Mean score among others.

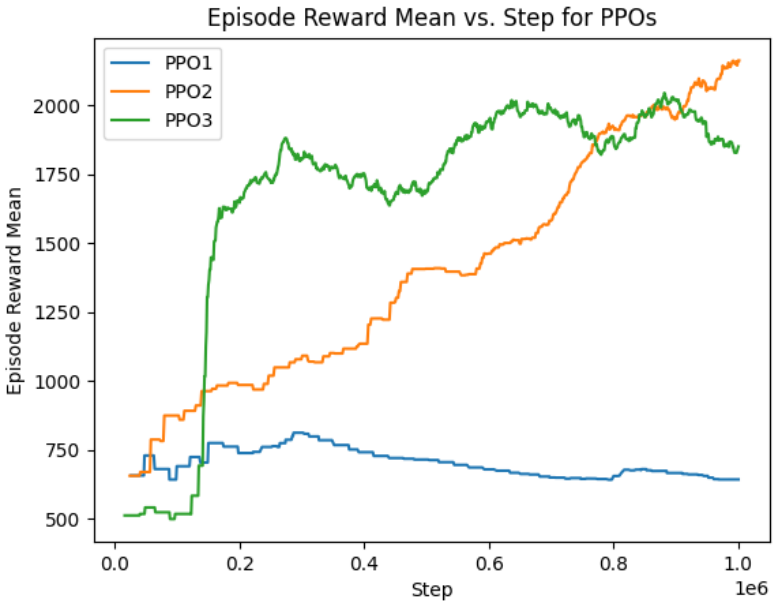

Figure 1. Episode Reward Mean vs Step for PPOs

2.2.2. PPOs Entropy Loss

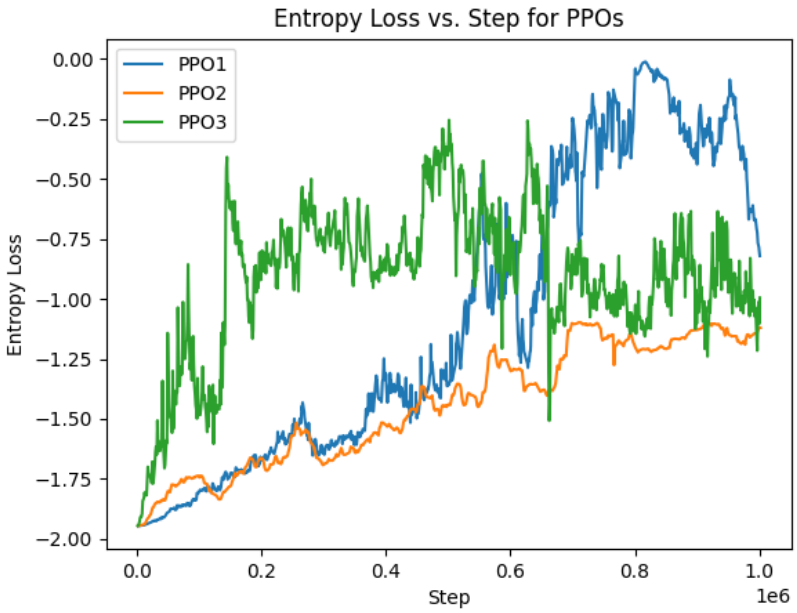

Figure 2. Entropy Loss vs Step for PPOs

2.2.3. DQNs Episode Reward Mean

As it can be seen from *Figure 3*, DQN3 has the best Episode Reward Mean score among others.

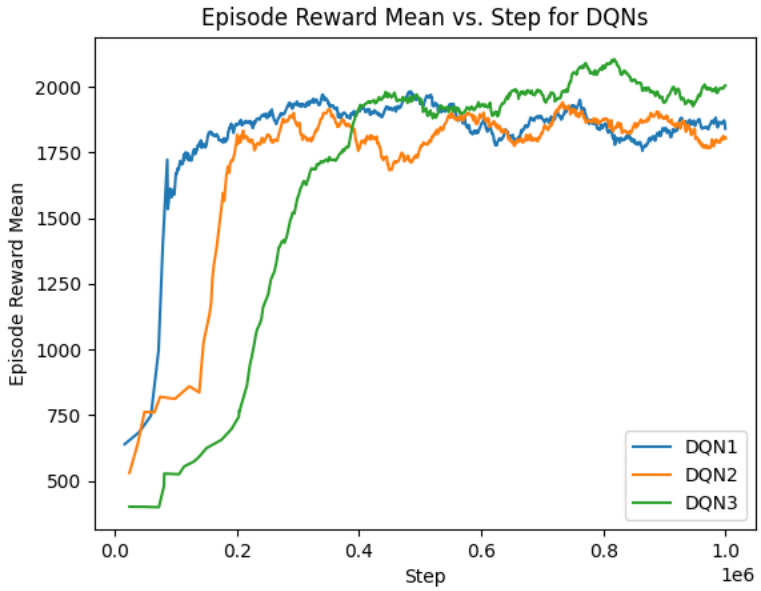

Figure 3. Episode Reward Mean vs Step for DQNs

2.2.4. DQNs Train Loss

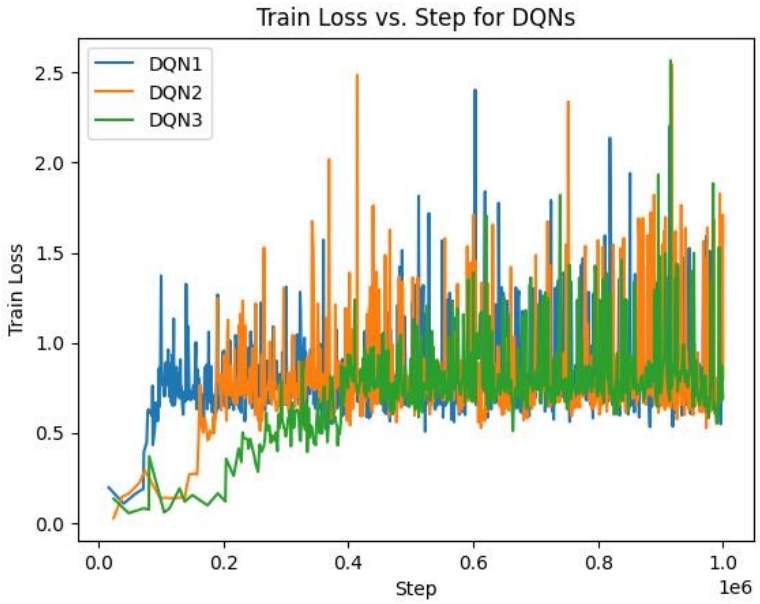

Figure 4. Train Loss vs Step for DQNs

2.2.5. PPO1 vs DQN1 (Preprocessing Type 0)

2.2.5.1. Episode Reward Mean

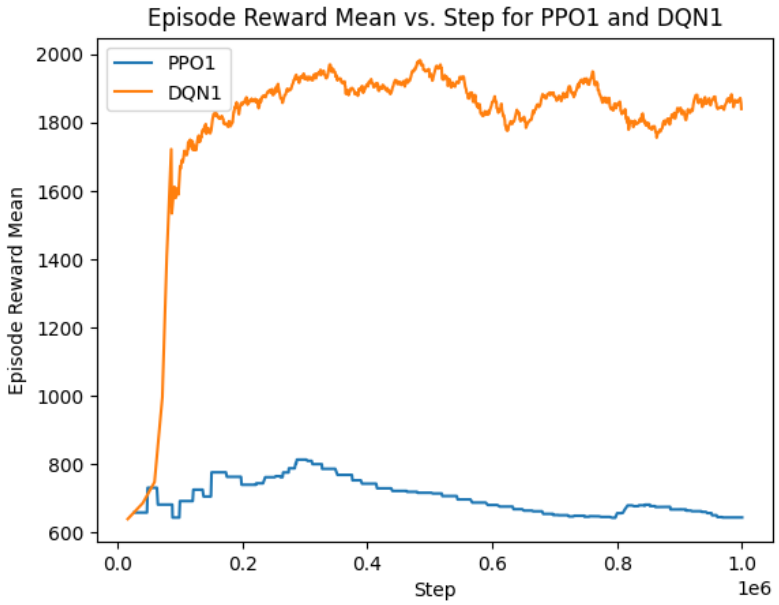

Figure 5. Episode Reward Mean vs Step for PPO1 and DQN1

2.2.5.2. Train Loss

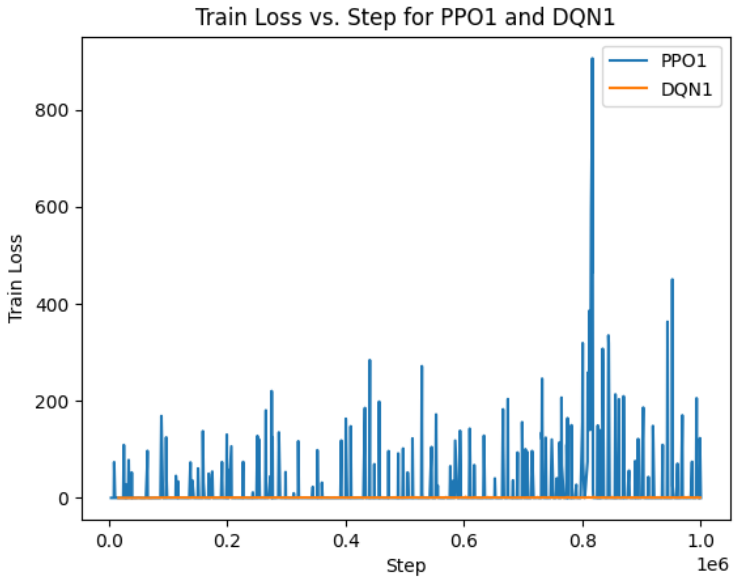

Figure 6. Train Loss vs Step for PPO1 and DQN1

2.2.5.3. Entropy Loss of and Train Loss

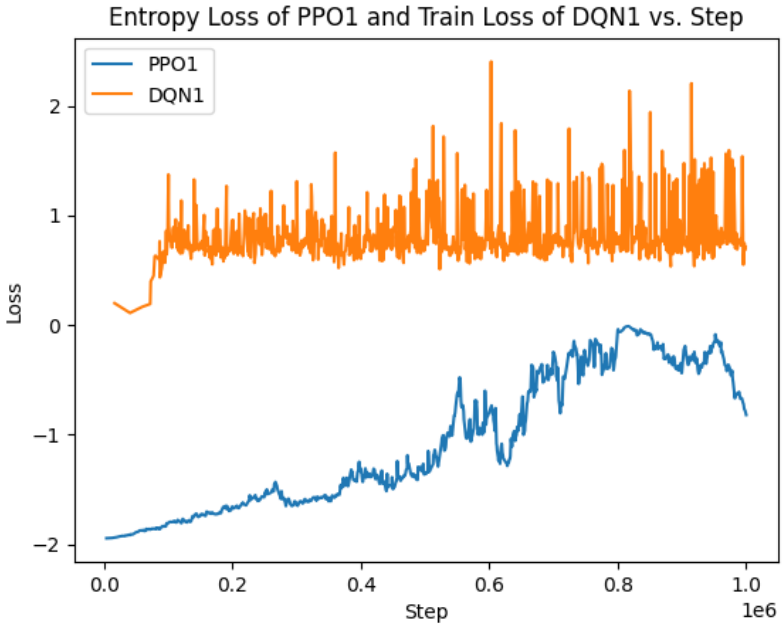

Figure 7. Entropy Loss of PPO1 and Train Loss of DQN1 vs Step

2.2.6. PPO2 vs DQN2 (Preprocessing Type 1)

2.2.6.1. Episode Reward Mean

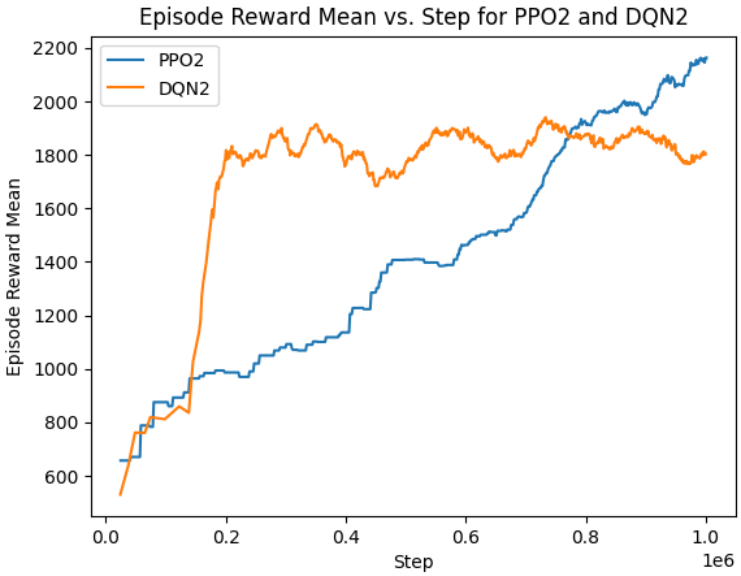

Figure 8. Episode Reward Mean vs Step for PPO2 and DQN2

2.2.6.2. Train Loss

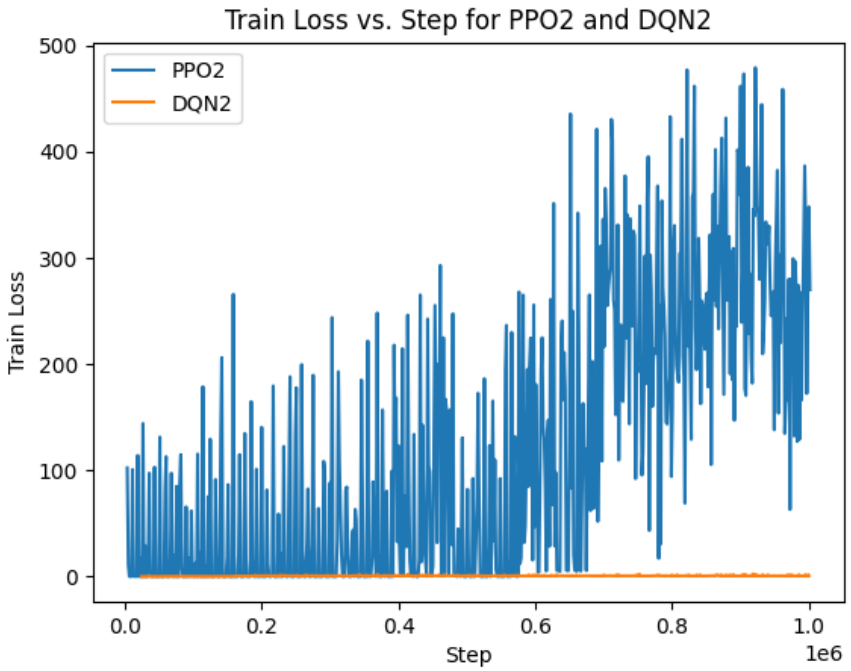

Figure 9. Train Loss vs Step for PPO2 and DQN2

2.2.6.3. Entropy Loss and Train Loss

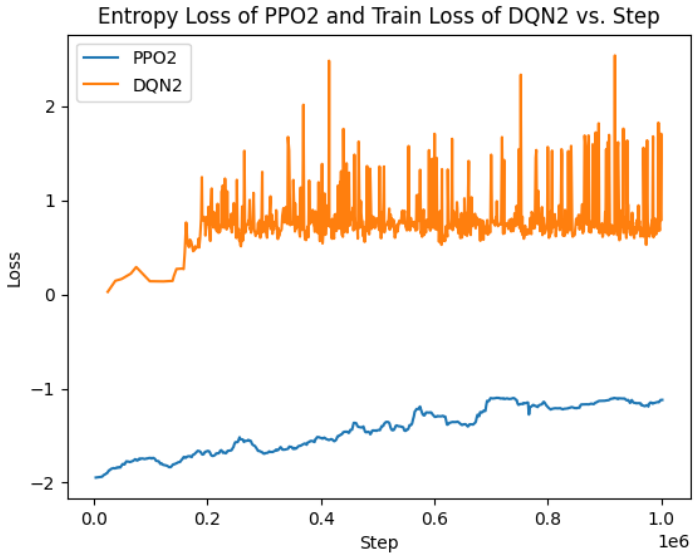

Figure 10. Entropy Loss of PPO2 and Train Loss of DQN2 vs Step

2.2.7. PPO3 vs DQN3 (Preprocessing Type 2)

2.2.7.1. Episode Reward Mean

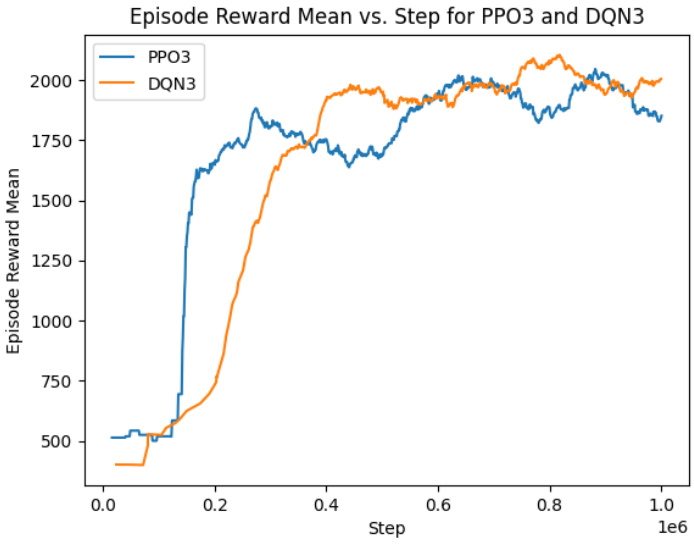

Figure 11. Episode Reward Mean vs Step for PPO3 and DQN3

2.2.7.2. Train Loss

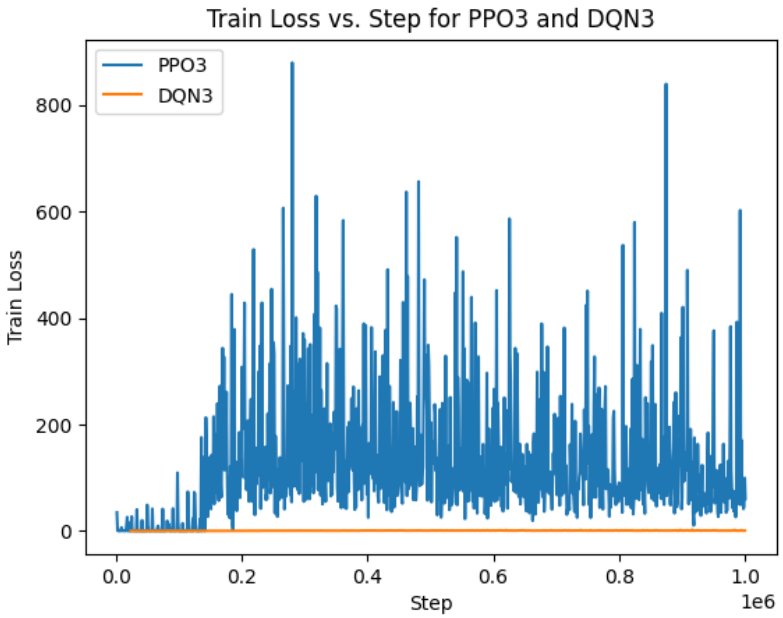

Figure 12. Train Loss vs Step for PPO2 and DQN2

2.2.7.3. Entropy Loss and Train Loss

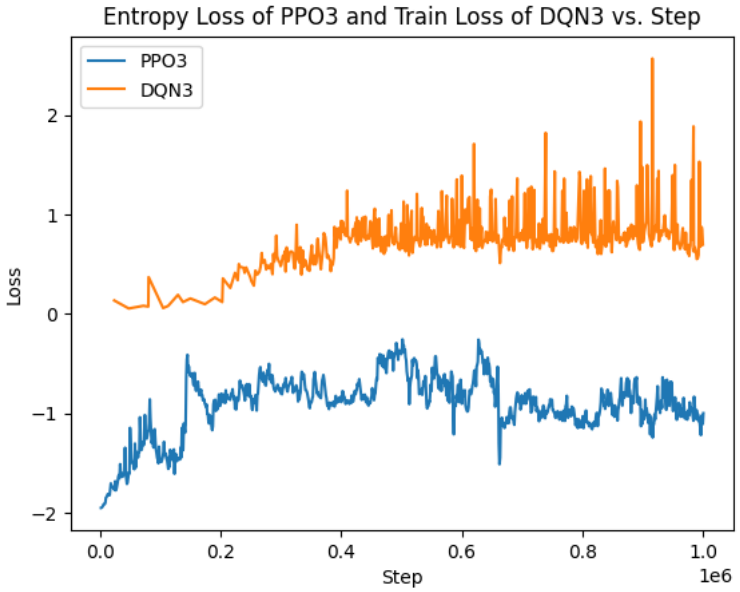

Figure 13. Entropy Loss of PPO3 and Train Loss of DQN3 vs Step

2.2.8. Train Loss for all models

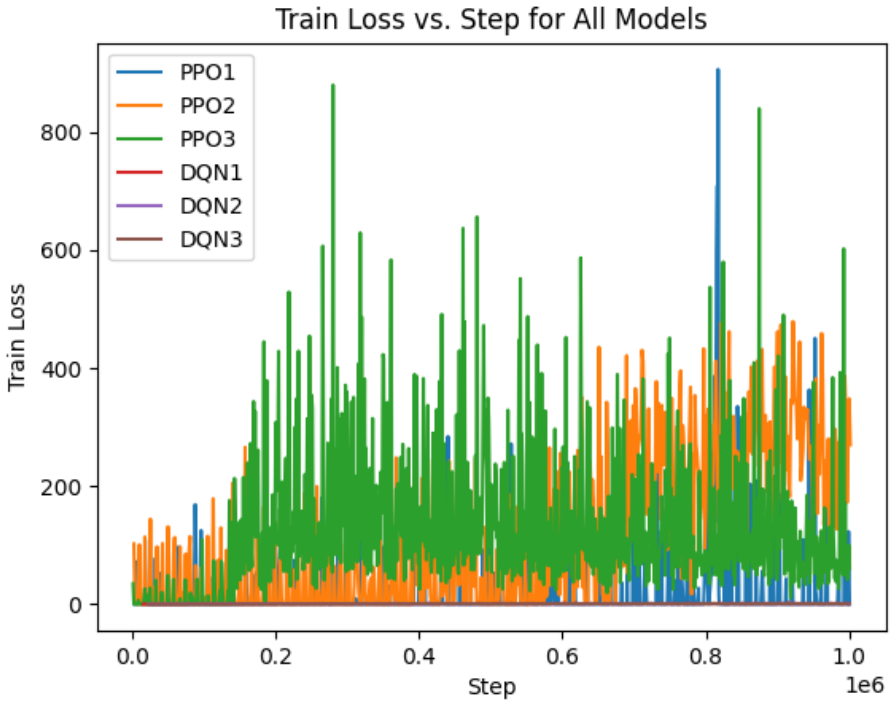

Figure 14. Train Loss for all models

2.2.9. Entropy Loss for PPOs and Train Loss for DQNs

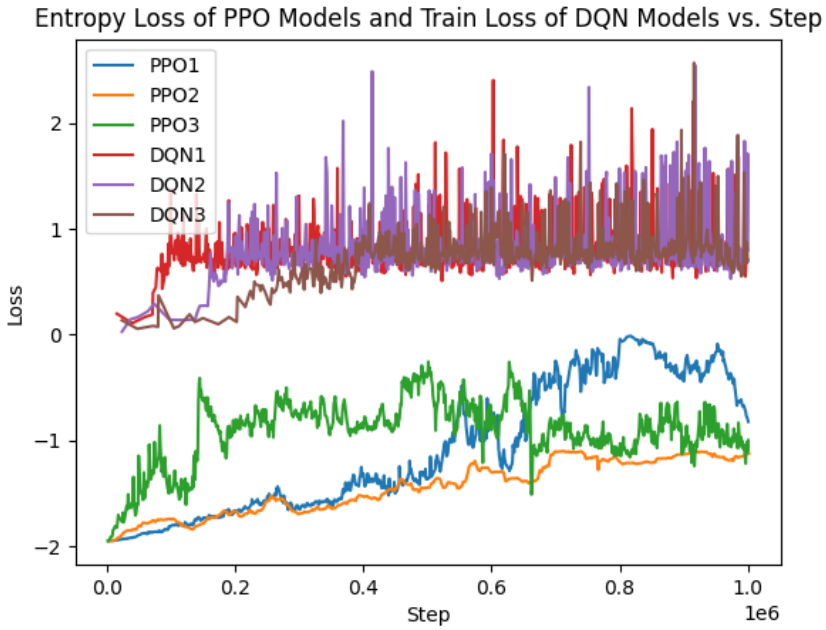

Figure 15. Entropy Loss for PPOs and Train Loss for DQNs

2.3. Best Model (PPO2) after 5 million steps

After observing the episode reward mean values for both of my 6 models, PPO2 has the highest episode reward mean values comparing to others. Therefore, I trained PPO2 for 5M steps.

Best Video Link (or click image below):
https://www.youtube.com/watch?v=R66Yy1wWQXc&ab_channel=DenizKarakay

2.3.1. Episode Reward Mean

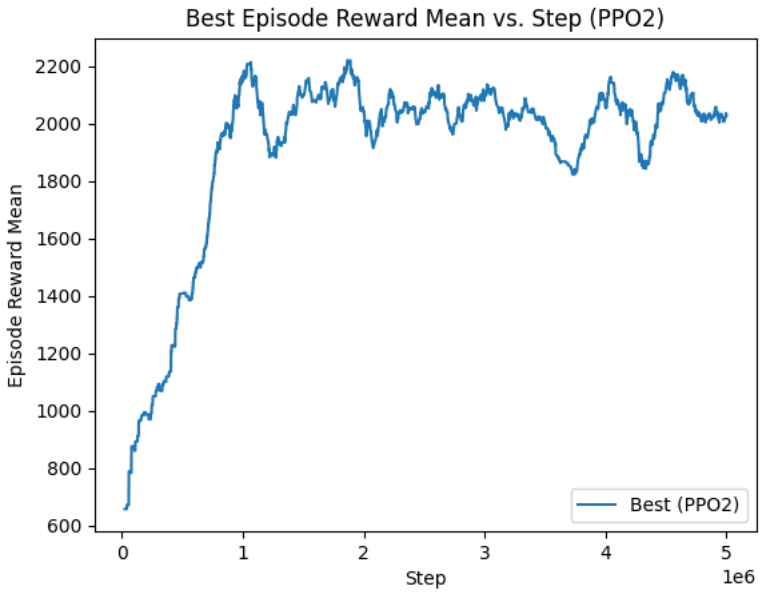

Figure 16. Best Model's Episode Reward Mean vs Step

2.3.2. Train Loss

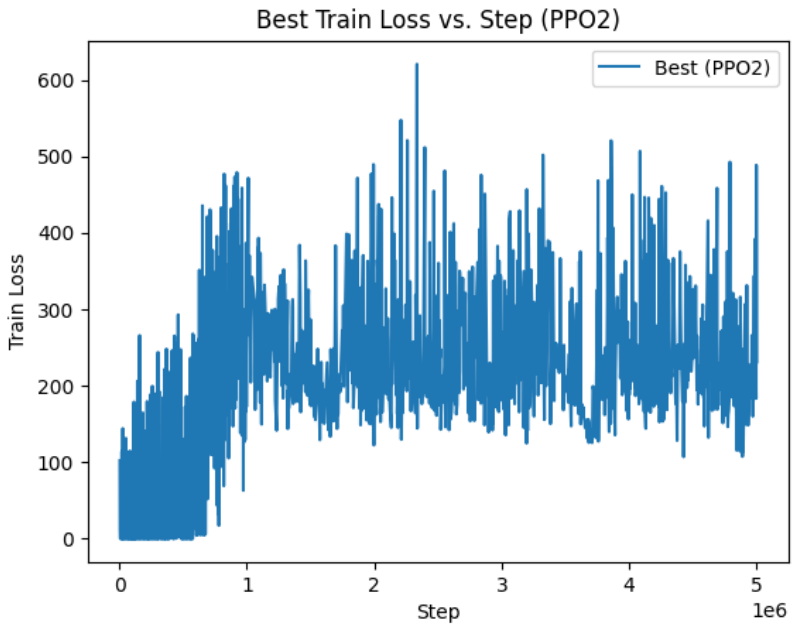

Figure 17. Best Model's Train Loss vs Step

2.3.3. Entropy Loss

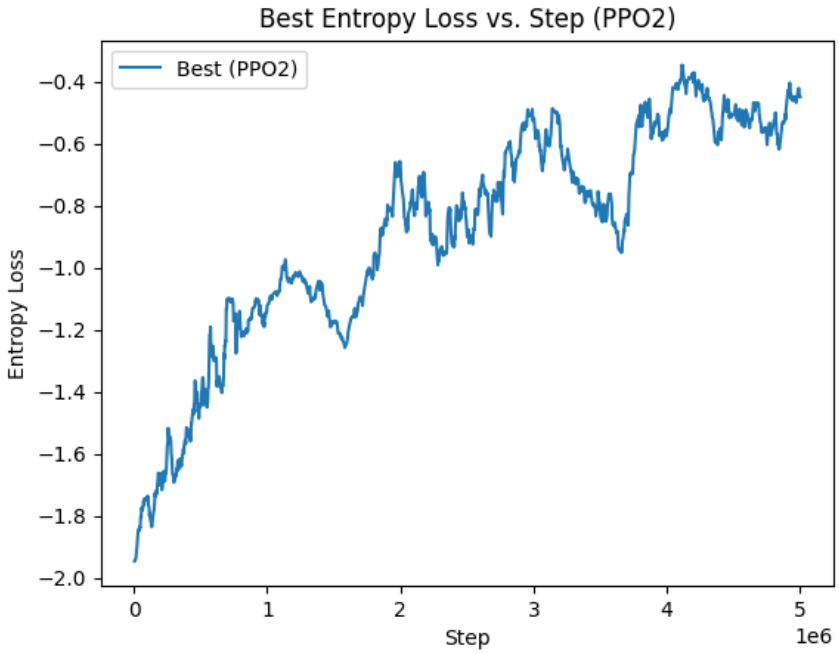

Figure 18. Best Entropy Loss' Train Loss vs Step

3. Discussions

3.1.

Table 4. Best Scores for all models based on different steps

Model	0 Step	10K Steps	100K Steps	500K Steps	1M Steps	5M Steps
PPO1	0	100	100	400	400	-
PPO2	0	200	200	1200	1300	17450
PPO3	0	0	400	1100	1200	-
DQN1	0	0	400	1000	1300	-
DQN2	0	100	200	700	800	-
DQN3	0	100	200	600	1200	-

Upon visually examining the learning progress of **PPO** and **DQN** algorithms, it is evident from *Table 4* and the gameplay of the models that learning occurs over time. Specifically, when considering longer pipes, I noticed that after approximately 500k steps, Mario consistently managed to jump over them. At the 100k step mark, Mario began to clear some pipes, but not the majority of them.

3.2.

PPO and DQN generally differ in their strengths and shortcomings. PPO is an algorithm that learns directly from the present policy because it is on-policy. It is a popular option for many RL jobs because of its stability and sample efficiency reputation. However, DQN is an off-policy algorithm that gains knowledge via a replay buffer of prior events. Although it might be more data-efficient, it might have stability problems.

By examining *Figures 1 and 3*, we can conclude that DQN-based models have a faster learning rate, which is reflected in the quicker saturation of episode reward mean values. While the values of each model are dependent on their respective hyperparameters (and not all of them were part of a controlled experiment), we can generally assert that DQN models learn faster.

3.3.

DQN employs an epsilon-greedy strategy in which the agent utilizes its present knowledge with a probability of 1-epsilon and engages in random activities with a likelihood of epsilon. Conversely, PPO balances exploration and exploitation while proactively maximizing the policy and using a substitute objective function to promote exploration. Because PPO directly optimizes the policy and stimulates exploration through the surrogate objective function, it is better than DQN at balancing exploration and exploitation. This results in more effective learning and more remarkable

performance in complicated situations.

3.4.

DQN often faces challenges when generalizing to new environments or unseen levels of a game due to its reliance on learning an accurate action-value function. In contrast, PPO is specifically designed to be more sample-efficient and stable compared to other policy gradient methods. It has demonstrated strong performance across a diverse array of tasks, encompassing continuous control and robotics. PPO's policy-based approach, which directly optimizes the policy instead of learning an action-value function, has the potential to generalize more effectively to new environments or unseen game levels. After personally observing the gameplay of my top-performing DQN and PPO models, I can attest to the superior performance of the PPO approach.

3.5.

The performance and learning speed of both PPO and DQN algorithms are heavily influenced by their hyperparameters. When examining the parameters in Table 2 for PPO, it becomes clear that the *learning rate* and *n_steps* are crucial for the algorithm's performance. Although I could not conduct a fully controlled experiment due to time and resource constraints, I observed that increasing the number of steps led to a higher episode reward mean. Additionally, raising the learning rates for both **PPO2** and **PPO3** resulted in improved outcomes. While I anticipated a less successful performance from MlpPolicy compared to CnnPolicy, I did observe a difference, which could be primarily attributed to changes in other parameters and the lack of a fully controlled experiment.

Considering *Table 3* for DQN, *exploration fraction, learning rate*, and *batch size* significantly impact the final result. Reducing the batch size generally led to better outcomes, although other variables were also adjusted simultaneously. Increasing the exploration fraction (like learning rate around ten times) yields better results, as demonstrated by the superior performance of **DQN3**. Raising the final epsilon value also enhances the results. As expected, I observed a less successful performance from MlpPolicy than CnnPolicy.

3.6.

In terms of computational complexity, PPO and DQN have different requirements. PPO is generally considered to be more computationally expensive than DQN, due to its use of a surrogate objective function and the need to perform multiple gradient updates per iteration. However, for our task, I observed that DQN, is more time demanding algorithm than PPO. For instance, I was able to train PPO Best under 8 hours whereas DQN best under 12 hours.

For our task DQN's higher computational cost may make it less practical for real-time applications or large-scale problems. However, for some applications DQN is better

although it took more time.

3.7.

When handling high-dimensional input states like images, the choice between MlpPolicy and CnnPolicy can greatly affect algorithm performance. MlpPolicy, a feedforward neural network with multiple layers, is less efficient for high-dimensional input states, as it treats each input feature independently, leading to increased parameters and computational complexity. Conversely, CnnPolicy is designed for high-dimensional input states, utilizing convolutional layers to capture local patterns and features within images. This allows CNNs to process and learn from high-dimensional data more efficiently and with fewer parameters. Additionally, CNNs handle translation invariance, recognizing features regardless of their position in the image.

In summary, CnnPolicy is better suited for tasks involving high-dimensional input states like images due to its efficiency and fewer required parameters. However, in my case, I observed better results with MlpPolicy in PPOs, likely because other hyperparameters were adjusted, compensating for the negative effects of MlpPolicy.

Appendix I

training.py – The Main code for training

```
#
# Created on Thu Jun 01 2023
#
# Deniz Karakay 2443307
#
# EE449 HW3 - Training
#


import gym_super_mario_bros
import numpy as np

from nes_py.wrappers import JoypadSpace
from gym_super_mario_bros.actions import SIMPLE_MOVEMENT
from gym.wrappers import GrayScaleObservation
from stable_baselines3 import PPO
from stable_baselines3.common.vec_env import VecFrameStack, DummyVecEnv, VecMonitor
from matplotlib import pyplot as plt
from stable_baselines3 import DQN
from utils import SaveOnBestTrainingRewardCallback, startGameModel, startGameRand


def create_environment(type, checkpoint):
    env = gym_super_mario_bros.make("SuperMarioBros-v0")
    env = JoypadSpace(env, SIMPLE_MOVEMENT)

    # 1st Preprocessing - Type 0 for PPO1 and DQN1
    if type == 0:
        env = GrayScaleObservation(env, keep_dim=True)
        env = DummyVecEnv([lambda: env])
        env = VecFrameStack(env, n_stack=4, channels_order="last")
        env = VecMonitor(env, f"{checkpoint}/TestMonitor")

    # 2nd Preprocessing - Type 1 for PPO2 and DQN2
    elif type == 1:
```

```
        env = GrayScaleObservation(env, keep_dim=False)
        env = DummyVecEnv([lambda: env])
        env = VecFrameStack(env, n_stack=2)
        env = VecMonitor(env, f"{checkpoint}/TestMonitor")

    # 3rd Preprocessing - Type 2 for PPO3 and DQN3
    elif type == 2:
        env = GrayScaleObservation(env, keep_dim=True)
        env = DummyVecEnv([lambda: env])
        env = VecFrameStack(env, n_stack=2, channels_order="first")
        env = VecMonitor(env, f"{checkpoint}/TestMonitor")
    return env


# Checkpoints
CHECKPOINT_DIR_PPO1 = "./train/PPO1/"
CHECKPOINT_DIR_PPO2 = "./train/PPO2/"
CHECKPOINT_DIR_PPO3 = "./train/PPO3/"
CHECKPOINT_DIR_DQN1 = "./train/DQN1/"
CHECKPOINT_DIR_DQN2 = "./train/DQN2/"
CHECKPOINT_DIR_DQN3 = "./train/DQN2_2/"
CHECKPOINT_DIR_DQN2 = "./train/DQN2_MLP_2/"
CHECKPOINT_DIR_DQN3 = "./train/DQN3/"
CHECKPOINT_DIR_DQN_BEST = "./train/DQN_BEST/"
CHECKPOINT_DIR_PPO_BEST = "./train/PPO_BEST/"

LOG_DIR = "./logs/"


# Train PPO1 - Default
env_ppo = create_environment(0, CHECKPOINT_DIR_PPO1)
callback_ppo = SaveOnBestTrainingRewardCallback(
    save_freq=10000, check_freq=1000, chk_dir=CHECKPOINT_DIR_PPO1
)
model_ppo = PPO(
    "CnnPolicy",
    env_ppo,
```

```
    verbose=1,
    tensorboard_log=LOG_DIR,
    learning_rate=0.000001,
    n_steps=512,
)
model_ppo.learn(total_timesteps=1000000, callback=callback_ppo, tb_log_name="PPO1")


# Train PPO2 - Env 1 - MLP
env_ppo2 = create_environment(1, CHECKPOINT_DIR_PPO2)
callback_ppo2 = SaveOnBestTrainingRewardCallback(
    save_freq=10000, check_freq=1000, chk_dir=CHECKPOINT_DIR_PPO2
)
model_ppo2 = PPO(
    "MlpPolicy",
    env_ppo2,
    verbose=1,
    tensorboard_log=LOG_DIR,
    learning_rate=0.000015,
    n_steps=1536,
)

model_ppo2.learn(total_timesteps=1000000, callback=callback_ppo2, tb_log_name="PPO2")


# Train PPO3 - Env 2 - CNN
env_ppo3 = create_environment(2, CHECKPOINT_DIR_PPO3)
callback_ppo3 = SaveOnBestTrainingRewardCallback(
    save_freq=10000, check_freq=1000, chk_dir=CHECKPOINT_DIR_PPO3
)
model_ppo3 = PPO(
    "CnnPolicy",
    env_ppo3,
    verbose=1,
    tensorboard_log=LOG_DIR,
    learning_rate=0.000005,
    n_steps=1024,
```

```
    ent_coef=0.01,
    gamma=0.99,
    n_epochs=8,
)
model_ppo3.learn(total_timesteps=1000000, callback=callback_ppo3, tb_log_name="PPO3")

# Train DQN1 - Default
env_dqn = create_environment(0, CHECKPOINT_DIR_DQN1)
callback_dqn = SaveOnBestTrainingRewardCallback(
    save_freq=10000, check_freq=1000, chk_dir=CHECKPOINT_DIR_DQN1
)
model_dqn = DQN(
    "CnnPolicy",
    env_dqn,
    batch_size=192,
    verbose=1,
    learning_starts=10000,
    learning_rate=5e-3,
    exploration_fraction=0.1,
    exploration_initial_eps=1.0,
    exploration_final_eps=0.1,
    train_freq=8,
    buffer_size=10000,
    tensorboard_log=LOG_DIR,
)
model_dqn.learn(total_timesteps=1000000, log_interval=1, callback=callback_dqn)

# Train DQN2 - Env 1 - MLP
env_dqn2 = create_environment(1, CHECKPOINT_DIR_DQN2)
callback_dqn2 = SaveOnBestTrainingRewardCallback(
    save_freq=10000, check_freq=1000, chk_dir=CHECKPOINT_DIR_DQN2
)
model_dqn2 = DQN(
    "MlpPolicy",
    env_dqn2,
    batch_size=150,
    verbose=1,
```

```
    learning_starts=10000,
    learning_rate=10e-3,
    exploration_fraction=0.2,
    exploration_initial_eps=1.0,
    exploration_final_eps=0.1,
    train_freq=4,
    buffer_size=10000,
    tensorboard_log=LOG_DIR,
)
model_dqn2.learn(
    total_timesteps=1000000,
    log_interval=1,
    callback=callback_dqn2,
    tb_log_name="DQN2",
)


# Train DQN3 - Env 2 - CNN
env_dqn3 = create_environment(2, CHECKPOINT_DIR_DQN3)
callback_dqn3 = SaveOnBestTrainingRewardCallback(
    save_freq=10000, check_freq=1000, chk_dir=CHECKPOINT_DIR_DQN3
)

model_dqn3 = DQN(
    "CnnPolicy",
    env_dqn3,
    batch_size=165,
    verbose=1,
    learning_starts=10000,
    learning_rate=20e-3,
    exploration_fraction=0.3,
    exploration_initial_eps=1.0,
    exploration_final_eps=0.15,
    train_freq=8,
    buffer_size=10000,
    tensorboard_log=LOG_DIR,
)
```

```
model_dqn3.learn(
    total_timesteps=1000000,
    log_interval=1,
    callback=callback_dqn3,
    tb_log_name="DQN3",
)


# Train PPO2 _best - Env 1 - MLP
env_ppo_best = create_environment(1, CHECKPOINT_DIR_PPO_BEST)
callback_ppo_best = SaveOnBestTrainingRewardCallback(
    save_freq=10000, check_freq=1000, chk_dir=CHECKPOINT_DIR_PPO_BEST
)
model_ppo_best = PPO(
    "MlpPolicy",
    env_ppo_best,
    verbose=1,
    tensorboard_log=LOG_DIR,
    learning_rate=0.000015,
    n_steps=1536,
)
model_ppo_best.learn(
    total_timesteps=1000000, callback=callback_ppo_best, tb_log_name="PPO_BEST"
)
```

Appendix II

plotting.py – For visualization of the plots, combining images etc.

```
#
# Created on Sat Jun 10 2023
#
# Deniz Karakay 2443307
#
# EE449 HW3 - Plotting
#


import pandas as pd
import matplotlib.pyplot as plt

PPOs = ["PPO1", "PPO2", "PPO3"]
DQNs = ["DQN1", "DQN2", "DQN3"]

best_name = ["PPO2_best"]

PPO_ELs = []
PPO_ERMs = []
PPO_TLs = []

DQN_ERMs = []
DQN_TLs = []


BEST_EL = pd.read_csv("csv/entropy_loss/" + best_name[0] + ".csv")
```

```
BEST_ERM = pd.read_csv("csv/ep_rew_mean/" + best_name[0] + ".csv")
BEST_TL = pd.read_csv("csv/train_loss/" + best_name[0] + ".csv")


for ppo in PPOs:
    df = pd.read_csv("csv/entropy_loss/" + ppo + ".csv")
    PPO_ELs.append(df)

for ppo in PPOs:
    df = pd.read_csv("csv/ep_rew_mean/" + ppo + ".csv")
    PPO_ERMs.append(df)

for ppo in PPOs:
    df = pd.read_csv("csv/train_loss/" + ppo + ".csv")
    PPO_TLs.append(df)

for dqn in DQNs:
    df = pd.read_csv("csv/ep_rew_mean/" + dqn + ".csv")
    DQN_ERMs.append(df)

for dqn in DQNs:
    df = pd.read_csv("csv/train_loss/" + dqn + ".csv")
    DQN_TLs.append(df)


# Plot episode reward mean for PPOs
plt.figure()
for i in range(len(PPO_ERMs)):
    plt.plot(PPO_ERMs[i]["Step"], PPO_ERMs[i]["Value"], label=PPOs[i])
plt.xlabel("Step")
plt.ylabel("Episode Reward Mean")
plt.title("Episode Reward Mean vs. Step for PPOs")
plt.legend()
plt.savefig("plots/ppo_ep_rew_mean.png")

# Plot entropy loss for PPOs
plt.figure()
```

```
for i in range(len(PPO_ELs)):
    plt.plot(PPO_ELs[i]["Step"], PPO_ELs[i]["Value"], label=PPOs[i])

plt.xlabel("Step")
plt.ylabel("Entropy Loss")
plt.title("Entropy Loss vs. Step for PPOs")
plt.legend()
plt.savefig("plots/ppo_entropy_loss.png")

# Plot episode reward mean for DQNs
plt.figure()
for i in range(len(DQN_ERMs)):
    plt.plot(DQN_ERMs[i]["Step"], DQN_ERMs[i]["Value"], label=DQNs[i])
plt.xlabel("Step")
plt.ylabel("Episode Reward Mean")
plt.title("Episode Reward Mean vs. Step for DQNs")
plt.legend()
plt.savefig("plots/dqn_ep_rew_mean.png")

# Plot train loss for DQNs
plt.figure()
for i in range(len(DQN_TLs)):
    plt.plot(DQN_TLs[i]["Step"], DQN_TLs[i]["Value"], label=DQNs[i])
plt.xlabel("Step")
plt.ylabel("Train Loss")
plt.title("Train Loss vs. Step for DQNs")
plt.legend()
plt.savefig("plots/dqn_train_loss.png")


# Plot episode reward mean for PPO1 and DQN1
plt.figure()
plt.plot(PPO_ERMs[0]["Step"], PPO_ERMs[0]["Value"], label="PPO1")
plt.plot(DQN_ERMs[0]["Step"], DQN_ERMs[0]["Value"], label="DQN1")
plt.xlabel("Step")
plt.ylabel("Episode Reward Mean")
plt.title("Episode Reward Mean vs. Step for PPO1 and DQN1")
```

```
plt.legend()
plt.savefig("plots/ppo1_dqn1_ep_rew_mean.png")

# Plot episode reward mean for PPO2 and DQN2
plt.figure()
plt.plot(PPO_ERMs[1]["Step"], PPO_ERMs[1]["Value"], label="PPO2")
plt.plot(DQN_ERMs[1]["Step"], DQN_ERMs[1]["Value"], label="DQN2")
plt.xlabel("Step")
plt.ylabel("Episode Reward Mean")
plt.title("Episode Reward Mean vs. Step for PPO2 and DQN2")
plt.legend()
plt.savefig("plots/ppo2_dqn2_ep_rew_mean.png")

# Plot episode reward mean for PPO3 and DQN3
plt.figure()
plt.plot(PPO_ERMs[2]["Step"], PPO_ERMs[2]["Value"], label="PPO3")
plt.plot(DQN_ERMs[2]["Step"], DQN_ERMs[2]["Value"], label="DQN3")
plt.xlabel("Step")
plt.ylabel("Episode Reward Mean")
plt.title("Episode Reward Mean vs. Step for PPO3 and DQN3")
plt.legend()
plt.savefig("plots/ppo3_dqn3_ep_rew_mean.png")

# Plot train loss for PPO1 and DQN1
plt.figure()
plt.plot(PPO_TLs[0]["Step"], PPO_TLs[0]["Value"], label="PPO1")
plt.plot(DQN_TLs[0]["Step"], DQN_TLs[0]["Value"], label="DQN1")
plt.xlabel("Step")
plt.ylabel("Train Loss")
plt.title("Train Loss vs. Step for PPO1 and DQN1")
plt.legend()
plt.savefig("plots/ppo1_dqn1_train_loss.png")

# Plot train loss for PPO2 and DQN2
plt.figure()
plt.plot(PPO_TLs[1]["Step"], PPO_TLs[1]["Value"], label="PPO2")
plt.plot(DQN_TLs[1]["Step"], DQN_TLs[1]["Value"], label="DQN2")
```

```
plt.xlabel("Step")
plt.ylabel("Train Loss")
plt.title("Train Loss vs. Step for PPO2 and DQN2")
plt.legend()
plt.savefig("plots/ppo2_dqn2_train_loss.png")

# Plot train loss for PPO3 and DQN3
plt.figure()
plt.plot(PPO_TLs[2]["Step"], PPO_TLs[2]["Value"], label="PPO3")
plt.plot(DQN_TLs[2]["Step"], DQN_TLs[2]["Value"], label="DQN3")
plt.xlabel("Step")
plt.ylabel("Train Loss")
plt.title("Train Loss vs. Step for PPO3 and DQN3")
plt.legend()
plt.savefig("plots/ppo3_dqn3_train_loss.png")


# Plot entropy loss for PPO1 and train loss for DQN1
plt.figure()
plt.plot(PPO_ELs[0]["Step"], PPO_ELs[0]["Value"], label="PPO1")
plt.plot(DQN_TLs[0]["Step"], DQN_TLs[0]["Value"], label="DQN1")
plt.xlabel("Step")
plt.ylabel("Loss")
plt.title("Entropy Loss of PPO1 and Train Loss of DQN1 vs. Step")
plt.legend()
plt.savefig("plots/ppo1_entropy_loss_dqn1_train_loss.png")

# Plot entropy loss for PPO2 and train loss for DQN2
plt.figure()
plt.plot(PPO_ELs[1]["Step"], PPO_ELs[1]["Value"], label="PPO2")
plt.plot(DQN_TLs[1]["Step"], DQN_TLs[1]["Value"], label="DQN2")
plt.xlabel("Step")
plt.ylabel("Loss")
plt.title("Entropy Loss of PPO2 and Train Loss of DQN2 vs. Step")
plt.legend()
plt.savefig("plots/ppo2_entropy_loss_dqn2_train_loss.png")
```

```
# Plot entropy loss for PPO3 and train loss for DQN3
plt.figure()
plt.plot(PPO_ELs[2]["Step"], PPO_ELs[2]["Value"], label="PPO3")
plt.plot(DQN_TLs[2]["Step"], DQN_TLs[2]["Value"], label="DQN3")
plt.xlabel("Step")
plt.ylabel("Loss")
plt.title("Entropy Loss of PPO3 and Train Loss of DQN3 vs. Step")
plt.legend()
plt.savefig("plots/ppo3_entropy_loss_dqn3_train_loss.png")

# Plot all train loss
plt.figure()
plt.plot(PPO_TLs[0]["Step"], PPO_TLs[0]["Value"], label="PPO1")
plt.plot(PPO_TLs[1]["Step"], PPO_TLs[1]["Value"], label="PPO2")
plt.plot(PPO_TLs[2]["Step"], PPO_TLs[2]["Value"], label="PPO3")
plt.plot(DQN_TLs[0]["Step"], DQN_TLs[0]["Value"], label="DQN1")
plt.plot(DQN_TLs[1]["Step"], DQN_TLs[1]["Value"], label="DQN2")
plt.plot(DQN_TLs[2]["Step"], DQN_TLs[2]["Value"], label="DQN3")
plt.xlabel("Step")
plt.ylabel("Train Loss")
plt.title("Train Loss vs. Step for All Models")
plt.legend()
plt.savefig("plots/all_train_loss.png")

# Plot entropy loss for all PPO models and train loss for all DQN models
plt.figure()
plt.plot(PPO_ELs[0]["Step"], PPO_ELs[0]["Value"], label="PPO1")
plt.plot(PPO_ELs[1]["Step"], PPO_ELs[1]["Value"], label="PPO2")
plt.plot(PPO_ELs[2]["Step"], PPO_ELs[2]["Value"], label="PPO3")
plt.plot(DQN_TLs[0]["Step"], DQN_TLs[0]["Value"], label="DQN1")
plt.plot(DQN_TLs[1]["Step"], DQN_TLs[1]["Value"], label="DQN2")
plt.plot(DQN_TLs[2]["Step"], DQN_TLs[2]["Value"], label="DQN3")
plt.xlabel("Step")
plt.ylabel("Loss")
plt.title("Entropy Loss of PPO Models and Train Loss of DQN Models vs. Step")
plt.legend()
plt.savefig("plots/all_entropy_loss_and_train_loss.png")
```

```
# Plot best episode reward mean
plt.figure()
plt.plot(BEST_ERM["Step"], BEST_ERM["Value"], label="Best (PPO2)")
plt.xlabel("Step")
plt.ylabel("Episode Reward Mean")
plt.title("Best Episode Reward Mean vs. Step (PPO2)")
plt.legend()
plt.savefig("plots/best_ep_rew_mean.png")

# Plot best train loss
plt.figure()
plt.plot(BEST_TL["Step"], BEST_TL["Value"], label="Best (PPO2)")
plt.xlabel("Step")
plt.ylabel("Train Loss")
plt.title("Best Train Loss vs. Step (PPO2)")
plt.legend()
plt.savefig("plots/best_train_loss.png")

# Plot best entropy loss
plt.figure()
plt.plot(BEST_EL["Step"], BEST_EL["Value"], label="Best (PPO2)")
plt.xlabel("Step")
plt.ylabel("Entropy Loss")
plt.title("Best Entropy Loss vs. Step (PPO2)")
plt.legend()
plt.savefig("plots/best_entropy_loss.png")
```

Appendix III

testing.py – For saving recording of best model, watching gameplays and determining the scores

```
#
# Created on Thu Jun 01 2023
#
# Deniz Karakay 2443307
#
# EE449 HW3 - Testing
#
```

```python
import gym_super_mario_bros
import numpy as np

from nes_py.wrappers import JoypadSpace
from gym_super_mario_bros.actions import SIMPLE_MOVEMENT
from gym.wrappers import GrayScaleObservation
from stable_baselines3 import PPO, DQN
from stable_baselines3.common.vec_env import VecFrameStack, DummyVecEnv, VecMonitor
from matplotlib import pyplot as plt

from utils import startGameModel, startGameRand, saveGameModel


def create_environment(type, checkpoint):
    env = gym_super_mario_bros.make("SuperMarioBros-v0")
    env = JoypadSpace(env, SIMPLE_MOVEMENT)
    if type == 0:
        env = GrayScaleObservation(env, keep_dim=True)
        env = DummyVecEnv([lambda: env])
        env = VecFrameStack(env, n_stack=4, channels_order="last")
        env = VecMonitor(env, f"{checkpoint}/TestMonitor")
    elif type == 1:
        env = GrayScaleObservation(env, keep_dim=False)
        env = DummyVecEnv([lambda: env])
        env = VecFrameStack(env, n_stack=2)
        env = VecMonitor(env, f"{checkpoint}/TestMonitor")
    elif type == 2:
        env = GrayScaleObservation(env, keep_dim=True)
        env = DummyVecEnv([lambda: env])
        env = VecFrameStack(env, n_stack=2, channels_order="first")
        env = VecMonitor(env, f"{checkpoint}/TestMonitor")
    return env


CHECKPOINT_DIR_PPO1 = "./train/PPO1/"
CHECKPOINT_DIR_PPO2 = "./train/PPO2/"
CHECKPOINT_DIR_PPO3 = "./train/PPO3/"
```

```
CHECKPOINT_DIR_DQN1 = "./train/DQN1/"
CHECKPOINT_DIR_DQN2 = "./train/DQN2_MLP_2/"
CHECKPOINT_DIR_DQN3 = "./train/DQN2_2/"
CHECKPOINT_DIR_DQN_BEST = "./train/DQN_BEST/"
CHECKPOINT_DIR_PPO_BEST = "./train/PPO_BEST/"

steps = [10000, 100000, 500000, 1000000]

# For testing the best models of PPO and DQN
# Print the scores of the best models
# I modified utils.py to save the scores of the best models

# Fpr PPO1
for s in steps:
    print(f"PPO1 - {s}")
    env = create_environment(0, CHECKPOINT_DIR_PPO1)
    model = PPO.load(f"{CHECKPOINT_DIR_PPO1}/models/iter_{s}")
    if s == 10000 or s == 100000:
        startGameModel(env, model, step_len=2500)
    else:
        startGameModel(env, model, step_len=10000)

# For PPO2
for s in steps:
    print(f"PPO2 - {s}")
    env = create_environment(1, CHECKPOINT_DIR_PPO2)
    model = PPO.load(f"{CHECKPOINT_DIR_PPO2}/models/iter_{s}")
    if s == 10000 or s == 100000:
        startGameModel(env, model, step_len=2500)
    else:
        startGameModel(env, model, step_len=10000)

# For PPO3
for s in steps:
    print(f"PPO3 - {s}")
    env = create_environment(2, CHECKPOINT_DIR_PPO3)
    model = PPO.load(f"{CHECKPOINT_DIR_PPO3}/models/iter_{s}")
```

```
    if s == 10000 or s == 100000:
        startGameModel(env, model, step_len=2500)
    else:
        startGameModel(env, model, step_len=10000)

# For DQN1
for s in steps:
    print(f"DQN1 - {s}")
    env = create_environment(0, CHECKPOINT_DIR_DQN1)
    model = DQN.load(f"{CHECKPOINT_DIR_DQN1}/models/iter_{s}")
    if s == 10000 or s == 100000:
        startGameModel(env, model, step_len=2500)
    else:
        startGameModel(env, model, step_len=10000)

# For DQN2
for s in steps:
    print(f"DQN2 - {s}")
    env = create_environment(1, CHECKPOINT_DIR_DQN2)
    model = DQN.load(f"{CHECKPOINT_DIR_DQN2}/models/iter_{s}")
    if s == 10000 or s == 100000:
        startGameModel(env, model, step_len=2500)
    else:
        startGameModel(env, model, step_len=10000)

# For DQN3
for s in steps:
    print(f"DQN3 - {s}")
    env = create_environment(2, CHECKPOINT_DIR_DQN3)
    model = DQN.load(f"{CHECKPOINT_DIR_DQN3}/models/iter_{s}")
    if s == 10000 or s == 100000:
        startGameModel(env, model, step_len=2500)
    else:
        startGameModel(env, model, step_len=10000)

# Record the video gameplay of the best model (PPO2 - 5M training steps)
env = create_environment(1, CHECKPOINT_DIR_PPO_BEST)
```

```
model = PPO.load(f"{CHECKPOINT_DIR_PPO_BEST}/best_model")
saveGameModel(env, model)
```