Aristotle University of Thessaloniki
School of Science
School of Informatics

Report in Deep Reinforcement Learning in Atari Games

Vasileios Asimakopoulos

3 Ιουνίου 2023

Περιεχόμενα

Κατάλογος σχημάτων

Code Listings

Κεφάλαιο 1

Introduction to the Project

1.1 Brief Description of the Code

Στην συγκεκριμένη εργασία χρησιμοποιήθηκαν για το Reinforcment Learning(RL), δύο βασικές βιβλιοθήκες:

Για το περιβάλλον του παιχνιδιού χρησιμοποιήθηκε η βιβλιοθήκη gymnasium της Farama Foundation, από την οποία επιλέχθηκε ένα atari παιχνίδι το Air-Raid. Για το Deep RL χρησιμοποιήθηκε η stable baselines3 στην οποία επιλέχθηκε να εκπαιδευτεί ο agent με την βοήθεια του αλγορίθμου PPO και A2C, για δύο policy, CNNPolicy και MlpPolicy. Αρχικά εκπαιδεύτηκαν τα μοντέλα με τις default τιμές των δύο policy και αργότερα προσθέθηκαν arguments στο policy. Και στις δύο περιπτώσεις των αλγορίθμων, στο CNNPolicy προστέθηκε μέσω της εντολής **policy kwargs** ένα CNN μοντέλο 1.1. Στην συνέχεια στο MlpPolicy προστέθηκε με τον ίδιο τρόπο ένα απλό νευρωνικό δίκτυο 1.2.

Code Listing 1.1: CNN Network

```
class CustomCNN(BaseFeaturesExtractor):
"""
Custom CNN architecture for feature extraction.
"""

def __init__(self, observation_space: gym.spaces.Discrete,
    features_dim: int = 128):
super().__init__(observation_space, features_dim)

# We assume CxHxW images (channels first)

n_input_channels = observation_space.shape[0]
self.cnn = nn.Sequential(
nn.Conv2d(n_input_channels, 128, kernel_size=4, stride=1,
    padding=0),
nn.ReLU(),
nn.Conv2d(128, 64, kernel_size=4, stride=1, padding=0),
nn.ReLU(),
nn.Flatten(),
)

# Compute shape by doing one forward pass
with torch.no_grad():
n_flatten = self.cnn(torch.as_tensor(observation_space.
    sample()[None]).float()).shape[1]
```

```
self.linear = nn.Sequential(nn.Linear(n_flatten,
    features_dim), nn.ReLU())

def forward(self, observations: torch.Tensor) -> torch.
    Tensor:
return self.linear(self.cnn(observations))


policy_kwargs = dict(
features_extractor_class=CustomCNN,
features_extractor_kwargs=dict(features_dim=128)
)
```

Code Listing 1.2: Neural Network

```
policy_kwargs = dict(activation_fn=torch.nn.ReLU,net_arch=
    dict(pi=[128, 64,32], vf=[128, 64,32]))
```

Ειδικότερα ο κώδικας που βλέπουμε παρακάτω 1.3 παρόλο που έχει το PPO σαν αλγόριθμο για τον agent παρέμεινε σταθέρος και στις 4 εκπαιδεύσεις(πέρα απο αυτά που αναφέρθηκαν παραπάνω). Το learning rate κρατήθηκε σταθερό με τιμή $10^{-}5$,εισήχθη η παράμετρος seed έτσι ώστε τα αποτελέσματα να είναι σταθερά κάθε φορά που έτρεχε το ίδιο μοντέλο.Τα steps κρατήθηκαν στα 200000, λόγω χαμηλής υπολογιστικής ισχύς. Έγινε μια εκπαίδευση στα 500000 αλλά δεν παρατηρήθηκε αύξηση, αντιθέτως τα αποτελέσματα ήταν πολύ χειρότερα απο ότι στα 200000 βήματα. Επίσης χρησιμοποιήθηκε το tensorboard για την παραγωγή διαγραμμάτων τα οποία θα σχολιαστούν στο επόμενο κεφάλαιο, ενώ τα enviroments που επιλέχθηκαν για να τρέξει το μοντέλο ήταν 6 λόγω χαμηλής χωρητικότητας της RAM(16gb).Επίσης τα enviroments χωρίστηκαν σε training τα οποία χρησιμοποιήθηκαν για την εκπαίδευση του μοντέλου και evaluation ώστε να υπάρχει ενα validaiton του μοντέλου(κάθε 1000 βήματα γινόταν το evaluation). Τέλος σώθηκαν τα μοντέλα για περαιτέρω fine-tuning αν χρειαστεί στο μέλλον, και παράχθει στο τέλος κάθε εκπάιδευσης ένα gif για να υπαρχει οπτική παρακολούθηση του μοντέλου.

Code Listing 1.3: Code of PPO algorithm

```
import os
import gymnasium as gym
import imageio
import numpy as np
import torch
import torch.nn as nn
from stable_baselines3 import PPO,A2C
from stable_baselines3.common.evaluation import
    evaluate_policy
from stable_baselines3.common.env_util import
    make_atari_env
from stable_baselines3.common.vec_env import VecFrameStack
from stable_baselines3.common.callbacks import CallbackList
    , CheckpointCallback, EvalCallback,
    StopTrainingOnNoModelImprovement
from stable_baselines3.common.torch_layers import
    BaseFeaturesExtractor
from stable_baselines3.common.vec_env import DummyVecEnv,
    SubprocVecEnv




if __name__ == "__main__":

        models_dir = "models/PPO_CNN_ORIGINAL"
        logdir = "logs"

        os.makedirs(models_dir, exist_ok=True)
        os.makedirs(logdir, exist_ok=True)



        # Check if GPU is available
        device = torch.device("cuda" if torch.cuda.
```

```
    is_available() else "cpu")
print("Device:", device)


class CustomCNN(BaseFeaturesExtractor):
        """
        Custom CNN architecture for feature
            extraction.
        """

        def __init__(self, observation_space: gym.
            spaces.Discrete, features_dim: int =
            128):
                super().__init__(observation_space,
                    features_dim)
                # We assume CxHxW images (channels
                    first)
                # Re-ordering will be done by pre-
                    preprocessing or wrapper
                n_input_channels =
                    observation_space.shape[0]
                self.cnn = nn.Sequential(
                nn.Conv2d(n_input_channels, 128,
                    kernel_size=4, stride=1, padding
                    =0),
                nn.ReLU(),
                nn.Conv2d(128, 64, kernel_size=4,
                    stride=1, padding=0),
                nn.ReLU(),
                nn.Flatten(),
                )

        # Compute shape by doing one forward pass
        with torch.no_grad():
                n_flatten = self.cnn(torch.
                    as_tensor(observation_space.
                    sample()[None]).float()).shape
                    [1]

                self.linear = nn.Sequential(nn.
                    Linear(n_flatten, features_dim),
                     nn.ReLU())

        def forward(self, observations: torch.
            Tensor) -> torch.Tensor:
                return self.linear(self.cnn(
                    observations))


policy_kwargs = dict(
features_extractor_class=CustomCNN,
```

```
features_extractor_kwargs=dict(features_dim=128)
)

# Create the base environment
base_env = make_atari_env("ALE/AirRaid-v5",
n_envs=8,
seed=21,
vec_env_cls=SubprocVecEnv
)

# Frame-stacking with 4 frames
train_env = VecFrameStack(base_env, n_stack=4)

# Separate evaluation env with the same base
    environment
eval_env = VecFrameStack(base_env, n_stack=4)


eval_callback = EvalCallback(eval_env,
best_model_save_path="./logs/best_model",
log_path="./logs/results",
eval_freq=1000,
verbose=1
)
checkpoint_callback = CheckpointCallback(save_freq
    =1000, save_path="./logs/")
# Create the callback list
callback = CallbackList([checkpoint_callback,
    eval_callback])

# Create the PPO agent with the custom network
model = PPO(
'CnnPolicy',
train_env,
# policy_kwargs=policy_kwargs,
verbose=1,
tensorboard_log=logdir,
device=device,
learning_rate=10e-5,
seed=21
)

TIMESTEPS = 200000

model.learn(total_timesteps=TIMESTEPS,
progress_bar=True,
callback=callback
)
model.save(f"{models_dir}/{TIMESTEPS}/")

print("Training␣done")
```

```
mean_reward, std_reward = evaluate_policy(model,
    eval_env , n_eval_episodes=10)
print("Mean␣Reward:", mean_reward)
print("Std␣Reward:", std_reward)

train_env.metadata['render_fps'] = 29

# Create gif
images = []
obs = model.env.reset()
img = model.env.render(mode="rgb_array")

for i in range(300):
        images.append(img)
        action, _ = model.predict(obs,
            deterministic=True)
        obs, _, _, _ = model.env.step(action)
        img = model.env.render(mode="rgb_array")

        # Save the GIF
        imageio.mimsave("air-raid_cnn_original.gif"
            , [np.array(img) for i, img in enumerate
            (images) if i % 2 == 0], duration=500)
```

Κεφάλαιο 2

Results of PPO and A2C

2.1 PPO Algorithm

Με τον αλγόριθμο PPO έγιναν 4 εκαπιδεύσεις:

1. Η εκπαίδευση πραγματοποιήθηκε με CNNPolicy, χωρίς ορίσματα (defaultCNNPolicy).

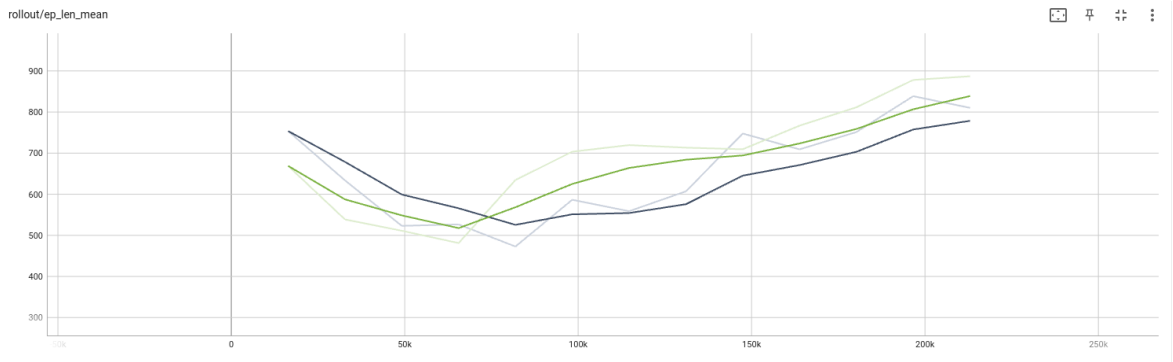
2. Η εκπαίδευση πραγματοποιήθηκε με CNNPolicy και με όρισμα το CNN μοντέλο (customCNNPolicy).

3. Η εκπαίδευση πραγματοποιήθηκε με MlpPolicy χωρίς ορίσματα (defaultMlpPolicy) .

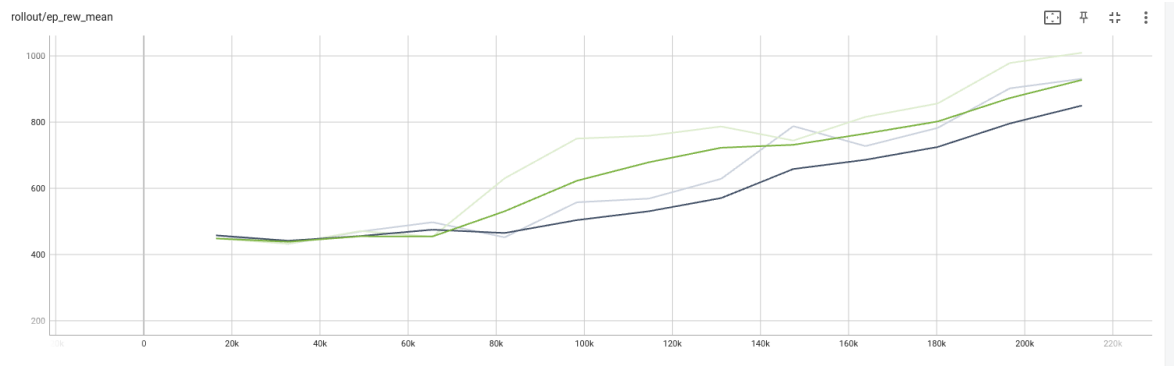
4. Η εκπαίδευση πραγματοποιήθηκε με MlpPolixy και με όρισμα το NN μοντέλο (customMlpPolicy).

2.1.1 Default CNNPolicy vs Custom CNNPolicy

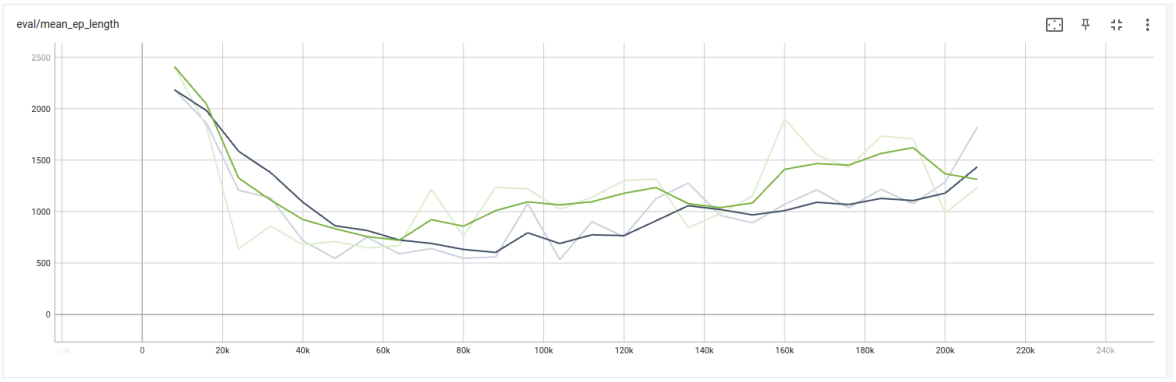
Στις δύο πρωτες εκπαιδεύσεις έχουν παραχθεί τα παρακάτω 5 διαγράμματα 2.1, 2.2, 2.3, 2.4 2.5. Το episode/evaluation length διάγραμμα μας εξηγεί πόση ώρα έτρεχε το παιχνιδι, δηλαδή μπορούμε να καταλάβουμε την διάρκεια που παρέμεινε το αεροπλάνο ζωντανό. Όσο αυξάνεται η διάρκεια του παιχνιδιού παρατηρούμε και άυξηση του reward που παίρνει σε κάθε επεισόδιο ή κατα την evaluation κατασταση. Το customCNNPolicy (η πράσινη γραμμή) παρόλο που στο κάθε επεισόδιο παραμένει σχεδόν πάντα πιο πάνω τόσο στο episode length (βλέπε 2.1) όσο και στο episode reward (βλέπε 2.2), στην evaluation φάση του μοντέλου και στα δύο διαγράμματα στο τέλος πέφτει, ενώ σε όλη την διάρκεια ήταν είτε στο ίδιο επίπεδο με το defaultCNNPolicy είτε λίγο πιο πάνω. Αυτό ίσως συμβαίνει λόγω της ευαισθησίας του CNN μοντέλου στις παραμέτρους που έχουμε βαλει κατα την εκπαίδευση ή απλά είναι ένα τυχαίο γεγονός και άν το συνεχίζαμε να ξανα περνούσε το defaultCNNPolicy. Η δεύτερη περίπτωση φαίνεται πιθανότερη παρατηρόντας το διάγραμμα με το train loss των δύο εκπαιδεύσεων 2.5. Επίσης στο defaultCNNPolicy έχουμε exploration-exploitation trade-off(εξήγηση παρακάτω).



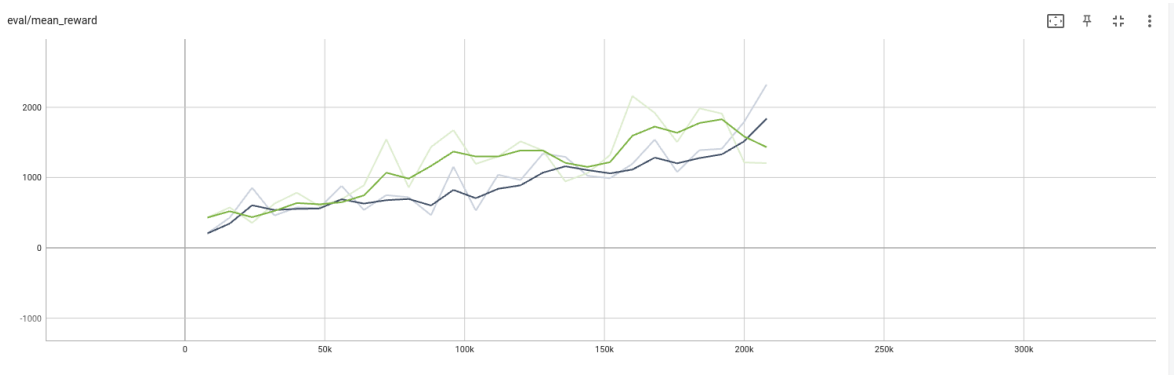
Σχήμα 2.1: Episode Length



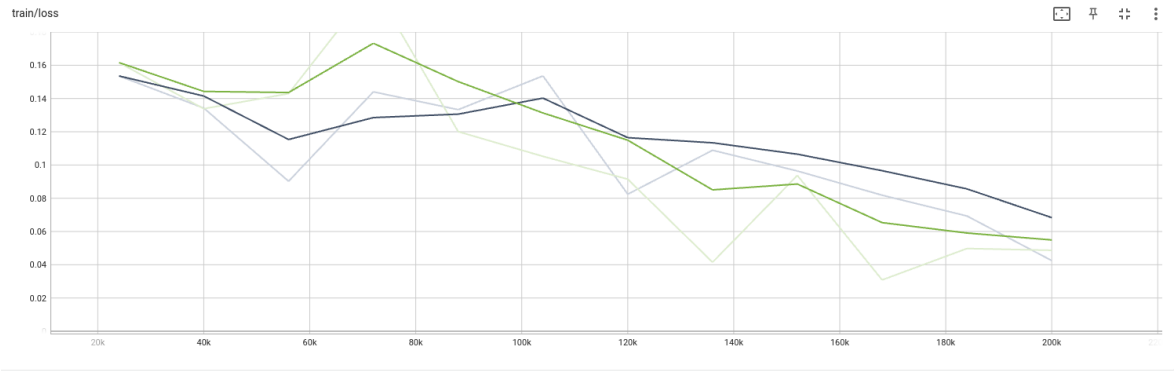
Σχήμα 2.2: Episode Reward



Σχήμα 2.3: Evaluation Length



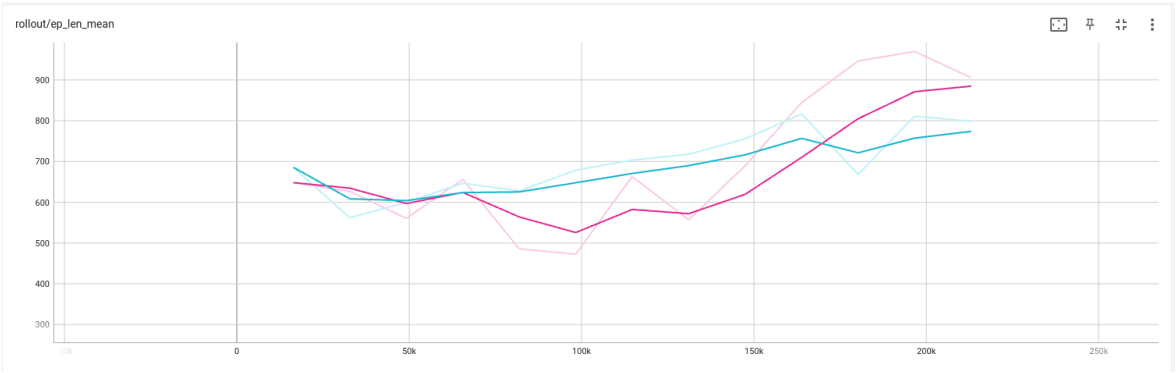
Σχήμα 2.4: Evaluation Reward



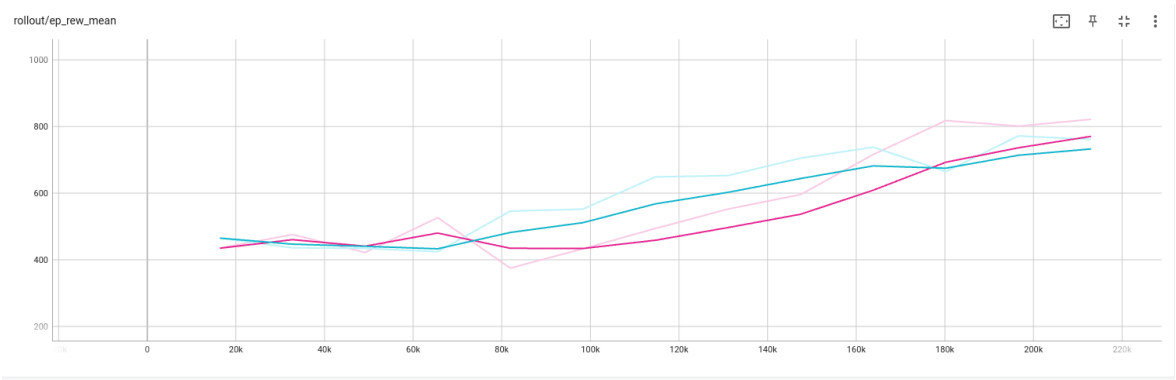
Σχήμα 2.5: Train loss

2.1.2 Default MlpPolicy vs Custom MlpPolicy

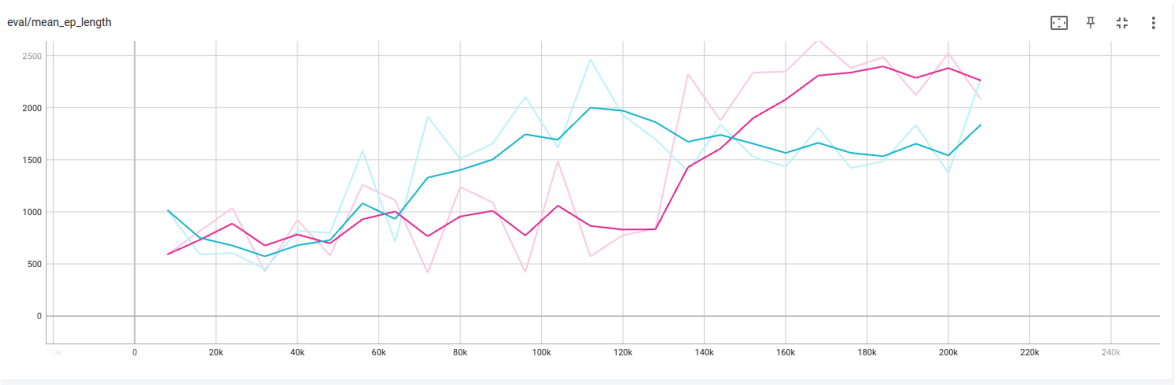
Στις δύο επόμενες εκπαιδεύσεις έχουν παραχθει τα παρακάτω 5 διαγράμματα 2.6, 2.7, 2.8, 2.9 2.10, όπως και πιο πανω. Η μπλέ γραμμή αντιστοιχεί στο CustomMlpPolicy και η ροζ στο DefaultMlpPolicy. Στην συγκεκριμένη περίπτωση παρατηρούμε οτι το DefaultMlpPolicy παρόλο που έχει μεγαλύτερο reward και length στα τελευταία βήματα απο ότι το custom στο evalutation πέφτει πολύ πιο κάτω απο το episode. Αυτό οφείλεται σε overfitting(ίσως reverse exploration-exploitation trade-off) του default αφου η μεγιστη τιμή στο episode reward ειναι σχεδόν 800 ενώ στο evaluation οριακά φτάνει στο 500. Μεγαλύτερη διαφορά παρατηρείται αντίστοιχα μεταξυ episode length και evaluation length. Από την άλλη το custom στο evalutation ανεβαινει σχεδόν στο διπλασιο το reward του σε σχέση με το episode, τότε έχουμε exploration-exploitation-trade-off. Δηλαδή κατα την διάρκεια του training episode ο agent δοκιμάζει μόνο actions χωρίς να δίνει έμφαση στα policies που το βοηθάνε να σκοτώσει τους εχθρούς και να μαζέψει ποντους οπότε παιρνει χαμηλο reward και δεν διαρκεί πολυ το παιχνιδι. Απο την άλλη κατα την evaluation κατάσταση χρησιμοποιείται μόνο το exploitation των policies και έχει συνέχως μεγάλα rewards, κατι που παρατηρουμε και στα episode/reward lengths διαγράμματα 2.6, 2.10



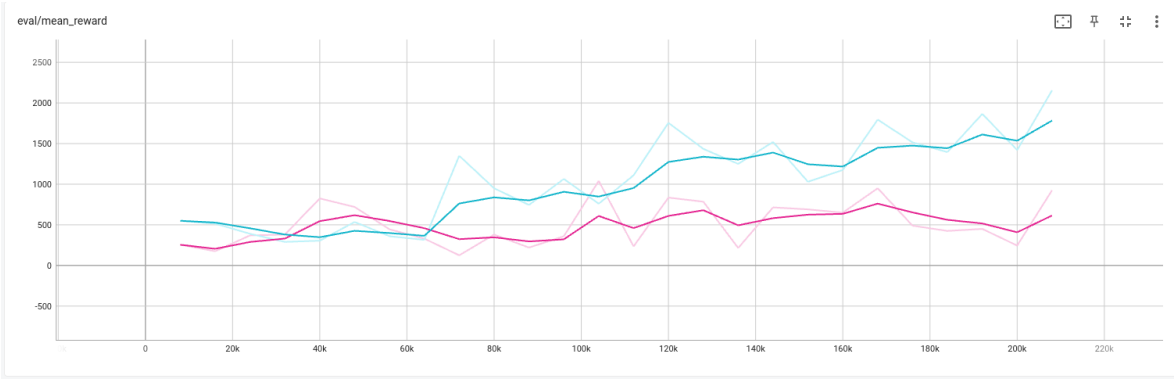
Σχήμα 2.6: Episode Length



Σχήμα 2.7: Episode Reward



Σχήμα 2.8: Evaluation Length



Σχήμα 2.9: Evaluation Reward

Σχήμα 2.10: Train loss

2.2 A2C Algorithm

Με τον αλγόριθμο A2C έγιναν 4 εκαπιδεύσεις:

1. Η εκπαίδευση πραγματοποιήθηκε με CNNPolicy, χωρίς ορίσματα (defaultCNNPolicy).

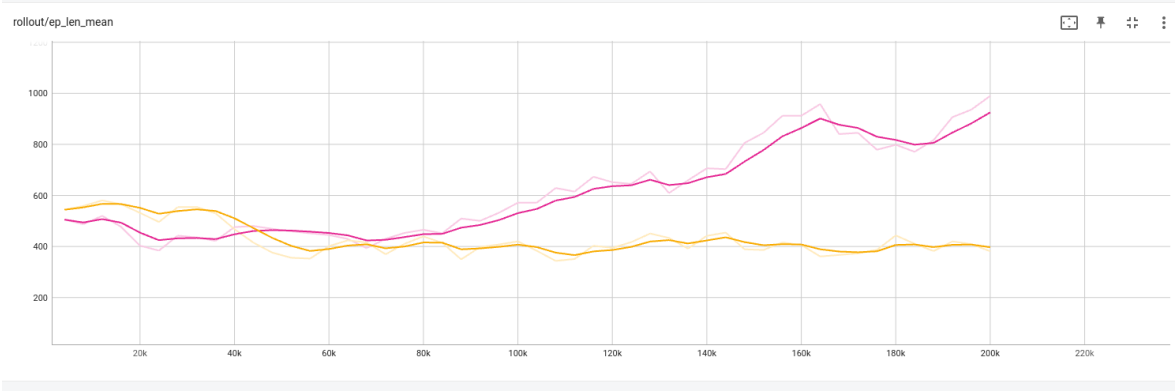
2. Η εκπαίδευση πραγματοποιήθηκε με CNNPolicy και με όρισμα το CNN μοντέλο (customCNNPolicy).

3. Η εκπαίδευση πραγματοποιήθηκε με MlpPolicy χωρίς ορίσματα (defaultMlpPolicy) .

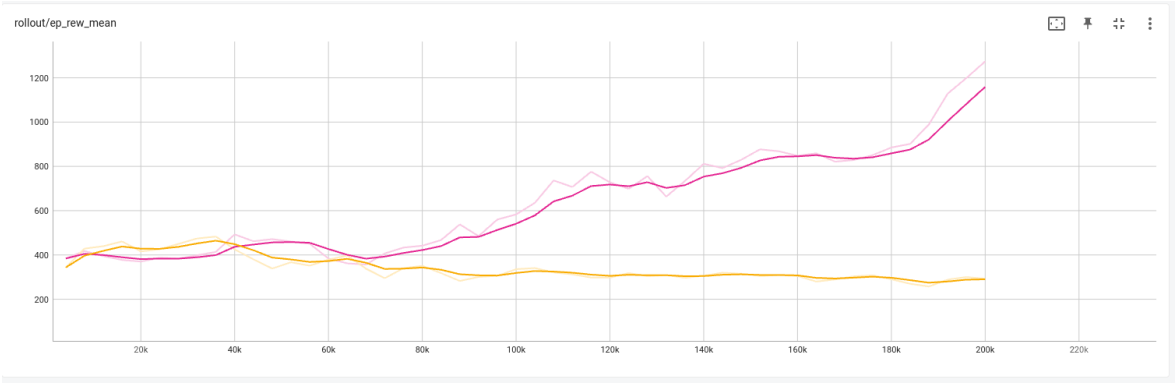
4. Η εκπαίδευση πραγματοποιήθηκε με MlpPolixy και με όρισμα το NN μοντέλο (customMlpPolicy).

2.2.1 Default CNNPolicy vs Custom CNNPolicy

Στις δύο πρωτες εκπαιδεύσεις έχουν παραχθεί τα παρακάτω 4 διαγράμματα 2.11, 2.12, 2.13, 2.14. Η πορτοκαλί γραμμή αντιστοιχεί στο CustomCNNPolicy και η ροζ στο DefaultCNNPolicy. Παρατηρείται και εδώ στο DefaultCNNPolicy μοντέλο exploration-exploitation trade-off, αφού το evaluation reward ειναι 2000 ενώ το episode reward είναι κάτω απο 1000. Απο την άλλη ενώ το customCNNPolicy έχει χαμηλή απόδοση παραμένει παρόμοια και στα δύο περιβάλλοντα. Το episode length του σε κάποιες στιγμές φτάνει να είναι ίδιο με του Default που αυτο μας δίνει να καταλαβουμε οτι ο agent έχει μάθει να προσπαθεί στο ελάχιστο για να μην χάσει.

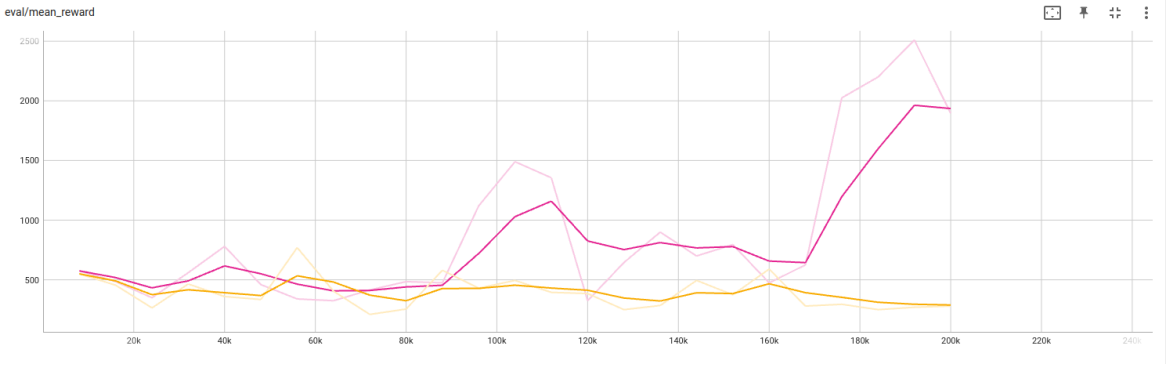


Σχήμα 2.11: Episode Length



Σχήμα 2.12: Episode Reward

Σχήμα 2.13: Evaluation Length



Σχήμα 2.14: Evaluation Reward

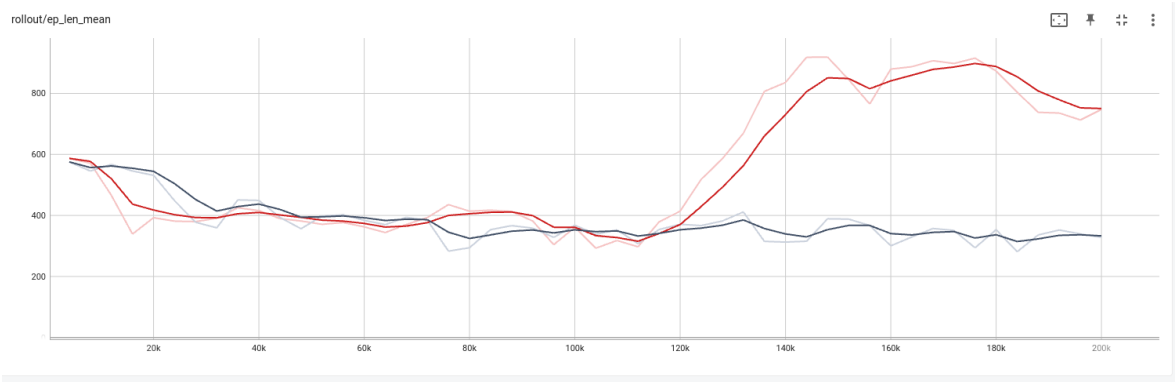
2.2.2 Default Mlpolicy vs Custom MlpPolicy

Στις δύο επόμενες εκπαιδεύσεις έχουν παραχθεί τα παρακάτω 4 διαγράμματα 2.15, 2.16, 2.17, 2.18, όπως πιο πάνω. Η μπλέ γραμμή αντιστοιχεί στο CustomMlp και η μάυρη αντιστοιχεί στο Default. Αυτό που παρατηρείται και στις δυο περιπτώσεις είναι οτι το length τόσο του evaluation όσο και του episode είναι παρα πολυ υψηλό σε σχεση με τα αντίστοιχα rewards. Αυτό μπορεί να οφείλεται Exploration vs. Exploitation, Suboptimal Policy, Sparse Rewards :

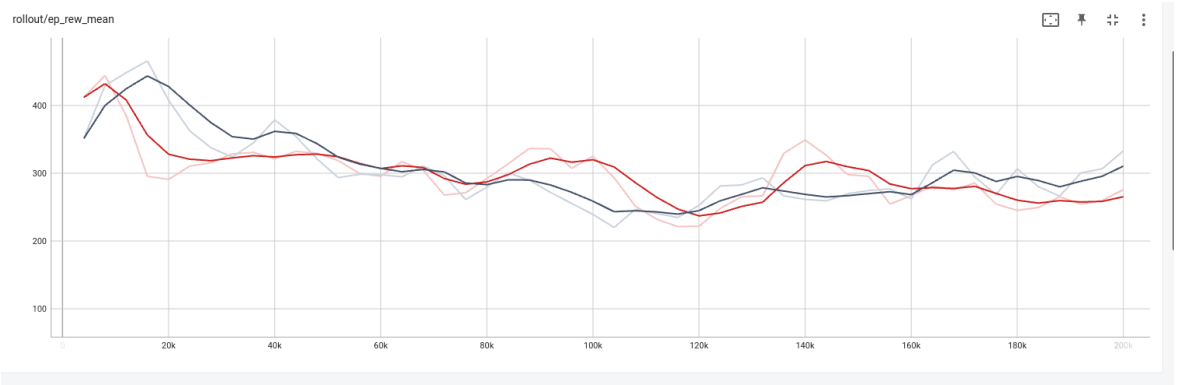
Ο αλγόριθμος A2C εξισορροπεί το exploration (δοκιμή νέων ενεργειών) με το exploitation (επιλογή ενεργειών που είναι πιθανό να αποφέρουν υψηλές ανταμοιβές). Εάν το length είναι σταθερά υψηλότερο από το reward, αυτό μπορεί να υποδεικνύει ότι ο agent εξερευνά κυρίως το περιβάλλον και δεν εκμεταλλεύεται ακόμη τη βέλτιστη στρατηγική για τη μεγιστοποίηση των ανταμοιβών, και επειδή σταματάμε στα 200000 βήματα που είναι σχετικά μικρή τιμή για RL, δεν ξέρουμε αν μετά ανέβει το reward

Επίσης το customCNN μοντέλο που περνάει στο CNNPolicy ενδέχεται να μην μαθαίνει αποτελεσματικά τα υποκείμενα μοτίβα του παιχνιδιού. Η αρχιτεκτονική του μοντέλου, οι υπερπαράμετροι ή η διαδικασία εκπαίδευσης μπορεί να προκαλούν το policy να παράγει ενέργειες που δεν ευθυγραμμίζονται με τη μεγιστοποίηση των rewards.

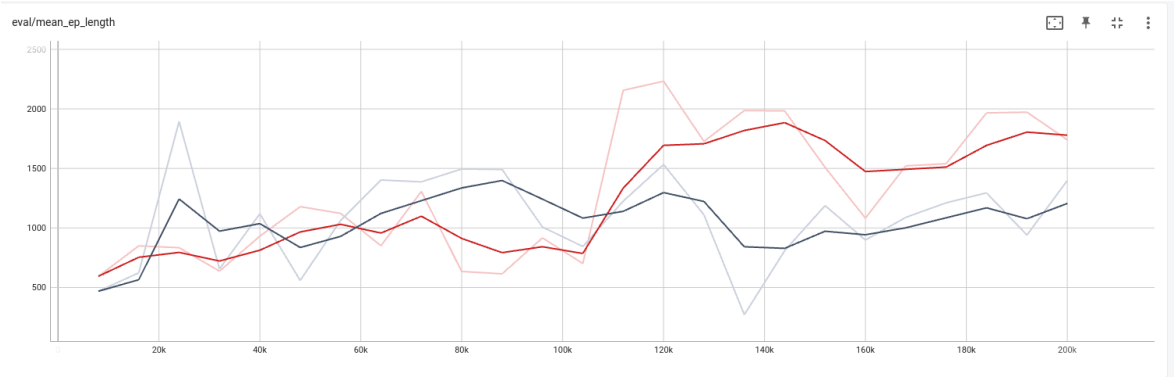
Τέλος τέτοιους είδους παιχνδίδια έχουν συνήθως sparse rewards signals, πράγμα που σημαίνει ότι ο agent λαμβάνει θετικά rewards σπάνια. Αυτό μπορεί να κάνει τη μάθηση πιο δύσκολη, καθώς ο agent μπορεί να δυσκολεύεται να συσχετίσει συγκεκριμένες ενέργειες με τις καθυστερημένα rewards. Σε τέτοιες περιπτώσεις, μπορεί να χρειαστεί περισσότερος χρόνος για να μάθει ο agent μια αποτελεσματική πολιτική που να επιτυγχάνει σταθερά υψηλά rewards.



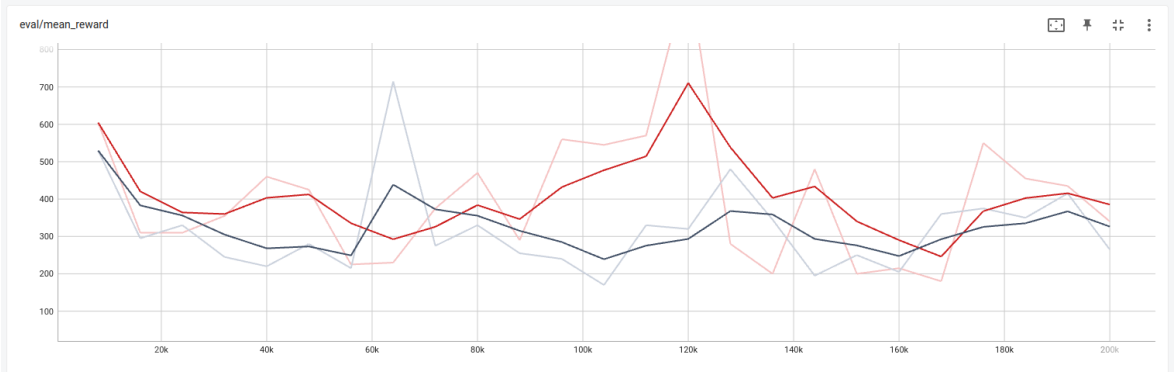
Σχήμα 2.15: Episode Length



Σχήμα 2.16: Episode Reward



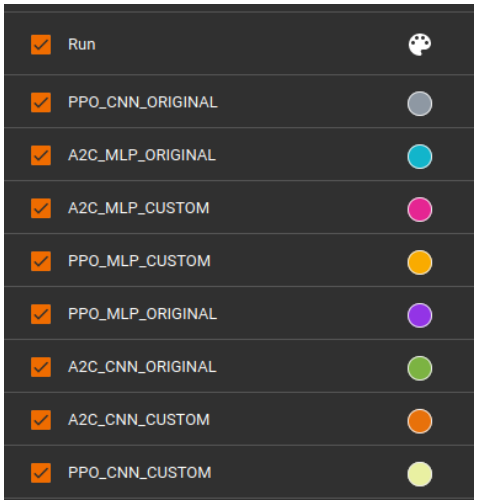
Σχήμα 2.17: Evaluation Length



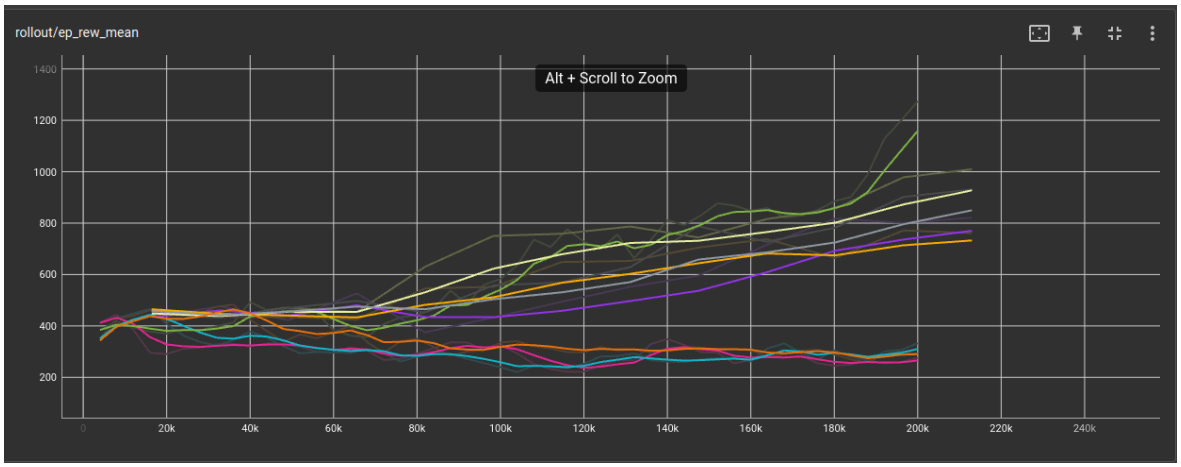
Σχήμα 2.18: Evaluation Reward

2.3 PPO vs A2C

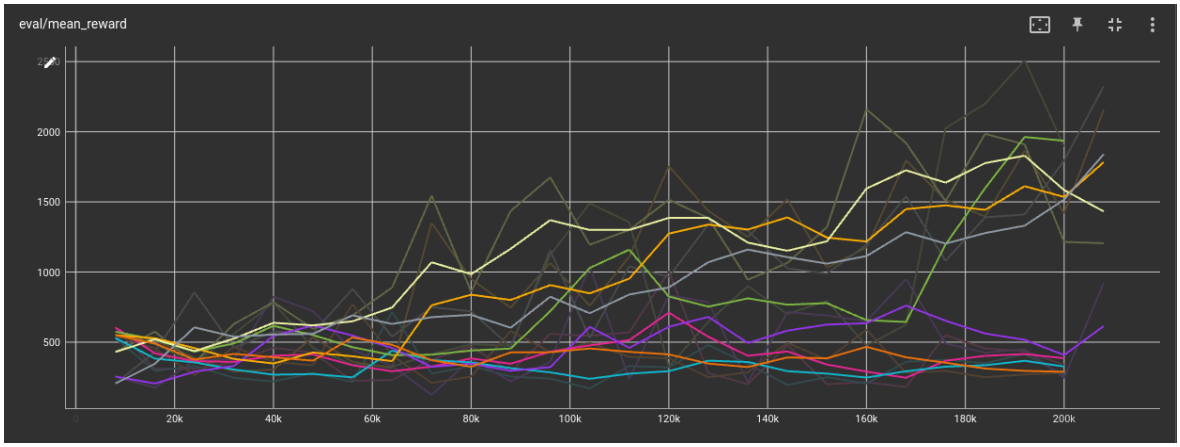
Στο τέλος συγκρίνουμε όλες τι εκπαιδεύσεις και παρατηρουμε στα δύο διαγράμμα 2.20, 2.21, ότι το A2C με defaultCNNPolicy έχει την καλύτερη απόδοση στο reward. Το PPO με customCNNPolicy στο episode reward διάγραμμα φαίνεται οτι έχει σταθερή αύξηση, ενώ στο evaluation reward διάγραμμα ξεκινάει στο τέλος να πέφτει. Τα δύο μοντέλα που έχουν σχετικά καλές αποδόσεις είναι τα PPO MLPCustomPolicy και PPO DefaultCnnPolicy, Τα υπόλοιπα μοντέλα είναι πιο ασταθή στην διάρκεια του χρόνου.



Σχήμα 2.19: Color of the Models



Σχήμα 2.20: Episode Reward



Σχήμα 2.21: Evaluation Reward