Comparison of Discrete and Continuous Action Spaces in Car Racing Environment

Piyush Malpure, * Mithulesh Ramkumar,† Rushikesh Pramod Deshmukh,‡
ppmalpure@wpi.edu,*mramkumar@wpi.edu,†rdeshmukh@wpi.edu,‡
Worcester Polytechnic Institute (WPI)
Worcester MA, USA

Abstract

CarRacing is a gym environment available in the OpenAI gym library. This environment has 96x96x3 pixel observation space and 3 actions: discrete steering (-1 for left steer and +1 for right steer), Acceleration and Breaking. This project worked with this environment to compare the effect of discrete, and continuous action spaces on the effectiveness of reinforcement learning. This project implemented two reinforcement learning algorithms: Deep Q Network Algorithm (DQN) and Continuous Proximal Policy Optimization (PPO) Reinforcement Learning (RL) Algorithm. It was found that PPO algorithm performed better than DQN algorithm in playing the Car Racing game by a huge margin. PPO based agent got an average reward of 217.38 over 10 episode while DQN based agent got an average of 92.

I. Introduction

Solving a reinforcement problem can be simplified to finding the optimal policy that would maximize the rewards for the problem. There are many algorithms that fall under the RL umbrella and they can all be broadly identified as either Model Free or Model based reinforcement learning. The term model in these distinctions represents the state transitions and the rewards of the agent and environment. Model based reinforcement learning is much simpler than model free reinforcement learning as the agent is able to predict what would happen for a range of possible actions and make decision based on that. Also, with model based reinforcement learning, the agent does not need to interact with the environment to train and hence does note require large training transition data sets. But with model free learning, we need to sample the environment by interacting with it to understand its working. These interactions are then stored in a buffer used later for training the RL agent.

In our project we took two different model free reinforcement algorithms and made comparisons based on their implementations. Under model free RL, they are split into two types : discrete action space based

RL and continuous action space based RL. The action space to interact with the environment is hybrid in nature; 1 discrete action i.e. steering (-1 or 1) and 2 continuous actions i.e Accelerations and Braking. In DQN learning we considered fixed discrete values for accelerations and braking and used only four possible actions to interact with environment. [steer, acc, braking] = [0 ,1.0, 0] or [1.0, 0.3 , 0] or [-1, 0.3, 0] or [0, 0, 0.8]. In PPO learning, we used log activation on the continuous value of steer to make it discrete. Recently, more advanced algorithms are being introduced to handle the hybrid action space like Parametrized-Deep Q Network learning (P-DQN) and hierarchical reinforcement learning. This project will explore and benchmark these algorithms in the future.

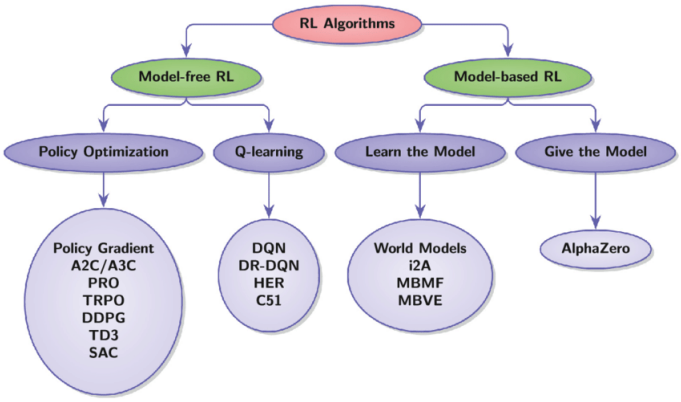

Fig. 1: Taxonomy of RL Methods

II. Background

The objective of CarRacing is to pilot a car through a randomly generated two-dimensional world of racetrack, grass, and boundaries, reaching the end of the track in as little time as possible. Predictably this encourages driving on the road and avoiding boundaries; from playing the game ourselves we know that the car tends to spin out of control on the grass, causing much time to be wasted during the jerky journey back to the road. The published implementation of CarRacing scores a completed run as follows: score(t) = 1000 - t/10 where t is the number of in-game frames it takes the car to reach the end of the track. The documentation claims that an algorithm which routinely achieves scores above 900, i.e. directs the car to the finish in at most 1000 frames, is sufficiently fit. The implementation is found here https://gym.openai.com/envs/CarRacing-v0/.

III. Methods

This project employed two strategies to train a agent that can play the Car Racing game. First strategy used discrete action space based DQN reinforcement learning to train the agent. discrete RL algorithms usually require less training episodes to train but might lack to control the system completely. Second

strategy used continuous action space based PPO reinforcement learning to train the RL agent. Continuous PPO algorithm requires a lot of training samples to give tangible results but has a better control over the action space. This project implemented these two algorithms to find the impact of choosing discrete action space verses choosing continuous action space. Also DQN is a value based reinforcement learning algorithm and PPO is policy based reinforcement learning algorithm. Comparison between these two approach to learning was interesting to observe and draw observations on.

A. Environment and Model Setup

The simulation environment used for this project is the CarRacing environment from Open AI's reinforcement learning gym system. This environment allows users to input different types of actions and obtain a result from it. The environment is also allowed to reset between each episode run. Due to this, multiple episode data can be obtained. The environment also allows users to render the environment to determine what the RL algorithm was able to solve and where it ran into problems. For the car racing problem, the environment has 96x96x3 pixel observation space and 3 actions: discrete steering (-1 for left steer and +1 for right steer), acceleration and breaking. Among the gym environments, this is the most easily implementable continuous action space due to not needing any physics engine like MujoCo to run the program.

B. DQN

DQN or Deep Q-Network is a type of value based RL algorithm that approximates a Q learning table/framework with the use of neural network. Q table consists of state action values where each state action pair is given a Q value. In normal Q learning, the number of states to consider and the number of action to consider make it very computationally intensive. For 10,000 states with 1000 action, this gives us a table of size 100,000. This increases drastically for more complex problems. Due to this Q learning is not feasible in real world environments. Therefore, DQNs are used. To utilize the power of deep neural networks, research was done to integrate the two. This was not easily realizable. Taking a Naive DQN with 3 convolution layer and 2 fully connected layers, researchers saw that implement a deep neural network was not a simple task. The results from that naive DQN and a linear DQN are given in 3. The Naive DQN was not able to get results due to overfitting and therefore more tuning was need. This was solved by Deepmind in their Atari paper as seen in [2]. They were able to overcome poor results of Naive DQN by four techniques:

1) Experience Replay
2) Target Network

3) Clipping Rewards
4) Skipping Frames

Experience replay were used to solve the overfitting issues that plagued the naive DQN. Due to how data was used, the model became unusable after overfitting as the variety of experiences reduces. This was solved by using mini batches to conduct the Q learning. These mini batches contained experience needed, like state, reward, action, etc to train and evaluate the deep neural network. Target network was updated in a set number of steps and were not updated every step like they were in deep neural network. This made the target network more stable and easier to train. Due to instability with the reward system, all positive reward were set to 1 and all negative rewards were set to -1. Skipping frames were also used to increase computational efficiency.

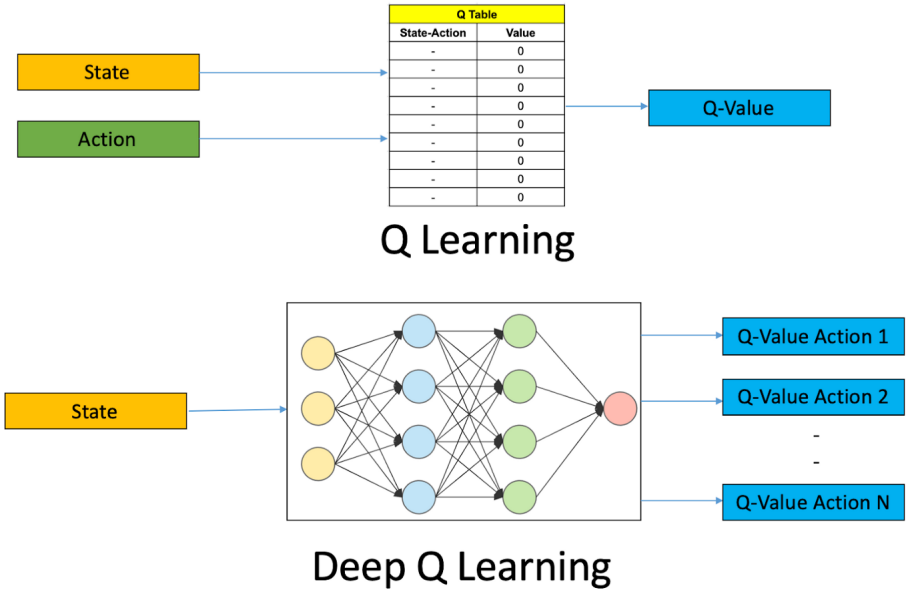

Fig. 2: DQN

	Breakout	R. Raid	Enduro	Sequest	S. Invaders
Naive DQN	**3.2**	1453.0	29.1	275.8	**302.0**
Linear	3.0	**2346.9**	**62.0**	**656.9**	301.3

Fig. 3: Results from Naive And Linear DQN

4 gives the equation that is used in DQN. This is a form of the bellman equation that is used to calculate the Q value for a state action pair.

1) Implementation of DQN: To implement DQN for car racing environment, custom DQN architecture was used. This was based on [3]. The architecture was defined as

$$Q(s,a) = E_{(s,a \to s',r) \sim \mathcal{H}} \left(r_{s,a} + \gamma \max_{a'} Q(s',a') \right)$$

Fig. 4: DQN Equation

Conv2d(1, 6)	(Kernel Size 4, Stride 4), RELU
Conv2d(6, 24)	(Kernel Size 4, Stride 1), RELU
MaxPool2d	(Kernel Size 2), FLATTEN
Linear	((9 x 9 x 24), 1000), RELU
Linear	(1000, 256), RELU
Linear	(256, 4)

TABLE I: DQN structure

The hyper parameters used for DQN are represented in table II

Hyperparameter	Value
Epsilon Start	1
Epsilon Goal	0.05
Epsilon Decay	30000
Episodes	225
Gamma	0.95
Learning Rate	0.0001
Training Frequency	4
Batch Size	64
Buffer Size	100000
Optimizer	RMSprop
Loss Criteria	Huber Loss

TABLE II: DQN Hyperparameters

C. Proximal Policy Optimization (PPO)

PPO [5] is a policy based algorithm which choose a greedy policy and has an improved performance in a smaller number of episodes and returns an approximately optimal policy. It is known for it's ease of hyperparameter tuning than the traditional reinforcement learning algorithms. PPO strikes a balance between episode sampling complexity, ease of implementation and ease of parameter tuning. It tries to update the policy at each step that minimizes the cost function while ensuring the deviation from the previous policy is relatively small. This is different from normal policy gradient, which keeps new and old polices close in parameter space. But, even small differences in the parameter space can have very

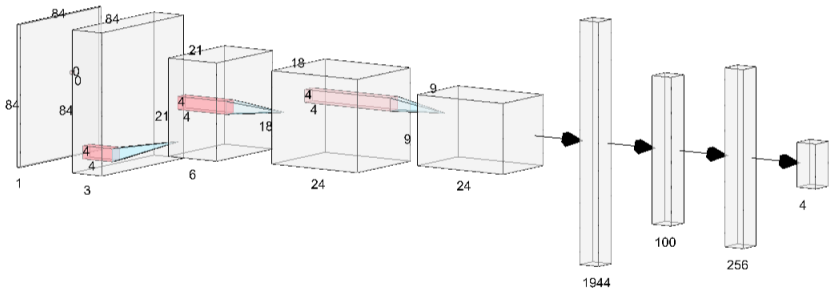

Fig. 5: Structure of DQN Network

large differences in performance so a single bad step can collapse the policy performance. This makes it dangerous to use large step sizes with vanilla policy gradients, thus also hurting sampling efficiency.

1) Algorithm: PPO has become the default learning algorithm at OpenAI because of its ease of use and good performancfe with optimal results.

There are two methods of PPO, one is PPO with Adaptive KL Penalty and the other is PPO with Clipped Objective. This paper uses PPO with clipped objective. In this algorithm two policy networks are used to compute the ratio which measures difference between the two policies, the current policy and the old policy. By importance sampling a new policy can be evaluated with samples collected from an older policy.

Algorithm for PPO with Clipped Objective is as shown below:

Input: initial policy parameters θ_0, clipping threshold ϵ

for $k = 0, 1, 2, \ldots$ **do**

Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$

Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm

Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}^{CLIP}(\theta)$$

by taking K steps of minibatch SGD (via Adam), where

$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_k} \left[\sum_{t=0}^{T} \left[\min(r_t(\theta)\hat{A}_t^{\pi_k}, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k}) \right] \right]$$

end for

Fig. 6: Algorithm for PPO with Clipped Objective

2) Model Parameters: The parameters chosen during the implementation are as shown in Table III.

Hyperparameter	Value
Learning Rate	**0.0003**, 0.0001, 0.00003, 0.000003
Image Stack	3
Discount Factor γ	**0.99**, 0.96, 0.98, 0.97
Maximum episodes	10000
Beta	**(0.9, 0.999)**, (0.9, 0.9), (0.8, 0.8)
Clip parameter	**0.2**, 0.8, 0.5
Loss Function	Mean squared error
Optimizer	Adam

TABLE III: Model Parameters

3) Structure of Actor-Critic Network: The Actor and the critic use a similar network structure. The network consists of a CNN backbone and two linear headers. The structure is shown in the Figure 7. The CNN takes the four stacked input images and extracts the features from them. The two headers are responsible for the beta heads of the PPO.

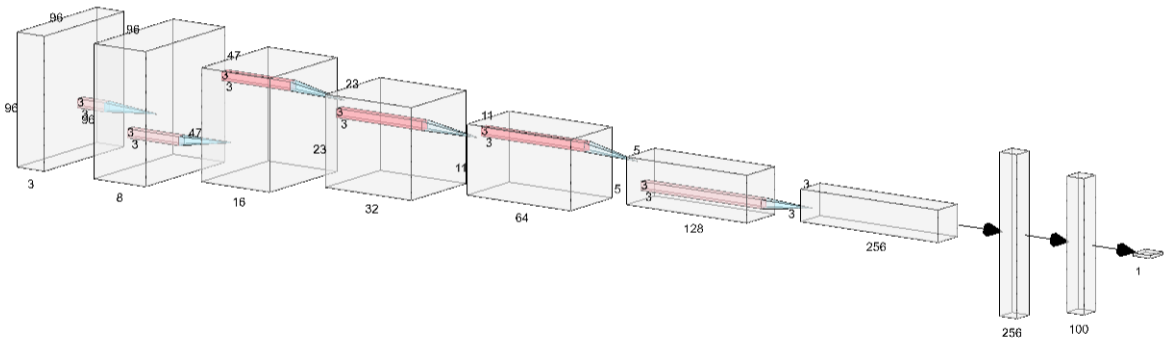

Fig. 7: Actor-Critic Neural network structure

The Structure parameters are also shown in the table IV. Where the first parameter gives the filter size and number of filters, second is the stride and fourth is the activation function.

conv1	(3, 3, 8), stride=2, ReLU
conv2	(3, 3, 16), stride=2, ReLU
conv3	(3, 3, 32), stride=2, ReLU
conv4	(3, 3, 64), stride=2, ReLU
conv5	(3, 3, 128), stride=2, ReLU
conv6	(3, 3, 256), stride=2, ReLU
beta1	Linear(256, 100), ReLU
beta2	Linear(256, 1), ReLU

TABLE IV: Actor structure

IV. Results

A. DQN

The DQN network was able to achieve a result of 92 over 10 episodes. There were predominantly two types of error in the model. The car was not able to steer tight corners. It was also going too fast and not be able to adjust to the changes in the road layout. Both these can be attributed to the limited action choices given to the model. Given that Car Racing environment is a continuous action space, only four control inputs were given to the DQN. In future works, this can be increased and more option for slower turning and slower travel in general can be given to the system.

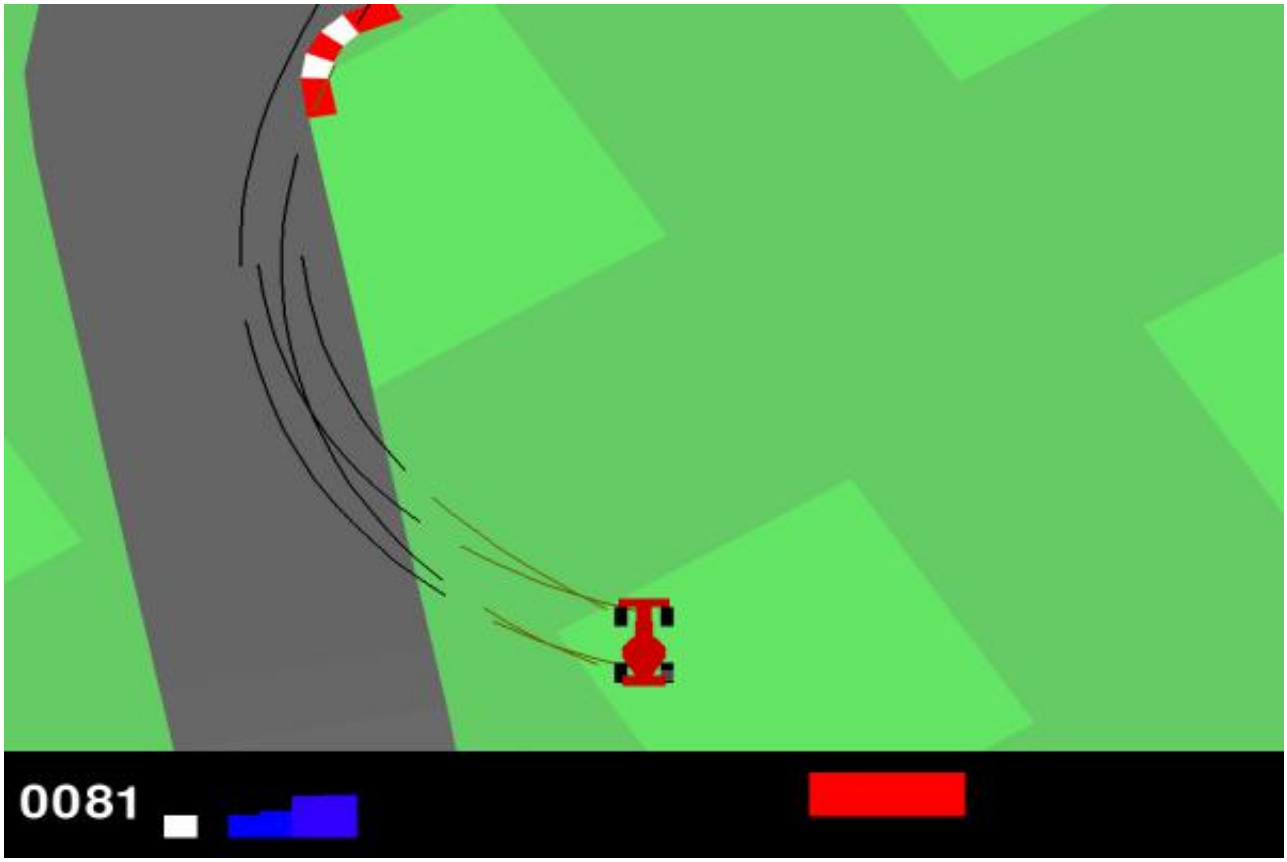

Fig. 8: Steering Error in DQN

The average reward from training graph are given in figure 9.

B. PPO

The PPO network was trained for 260 episodes using learning rate of 0.0003, discount factor γ as 0.99, reward clipping parameter of 0.2 with adam as the optimiser and mean squared error as the loss function. It was trained on a system using NVIDIA GTX 1050 for a period of 4 hours.

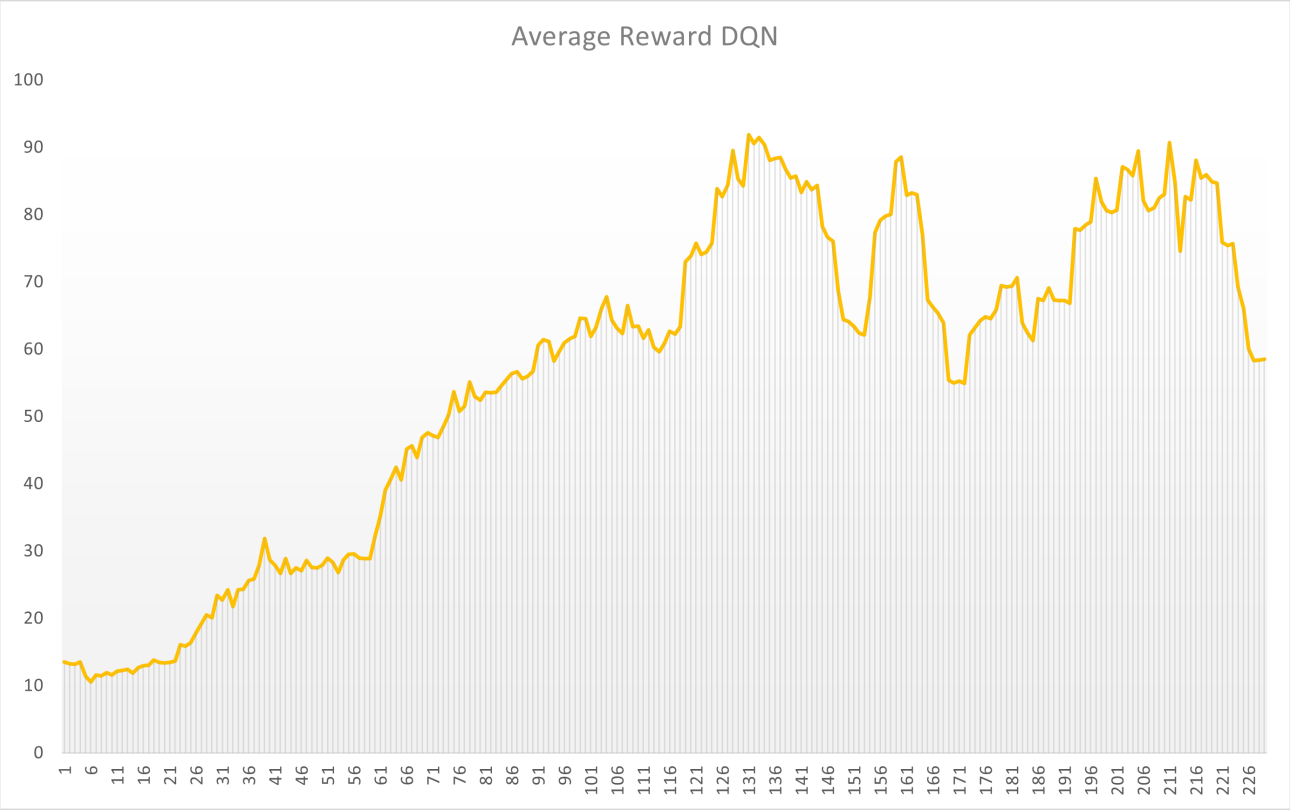

Fig. 9: Average Training Reward for DQN

The model was able to attain a good performance with an average score of 300 for training and 217.38 for testing calculated over 10 episodes.

The model is able to keep the car on the track, it turns properly and also has sufficient acceleration.

The figure 10below shows the reward versus the number of episodes during the training stage. The figure 11 shows the car rendered during the testing stages.

V. Discussion

As seen in the results it is clear that PPO implemented in continuous action space performs better than DQN implemented on a discrete space. PPO is a very robust algorithm and is more efficient as training advances, showing a very good performance in the environment. The car successfully manages to stay in the given track with a good acceleration using the trained agent.

Discrete action space makes the problem easier to implement but introduces certain challenges to the agent to learn and overcome problems of sliding and slipping as seen in the results. Reasons for failure of the DQN implemented in discrete space is mainly due to the limited number actions given reducing its degree of freedom. DQN is also highly dependent on fine tuning and is very sensitive to small changes which causes the model to fail.

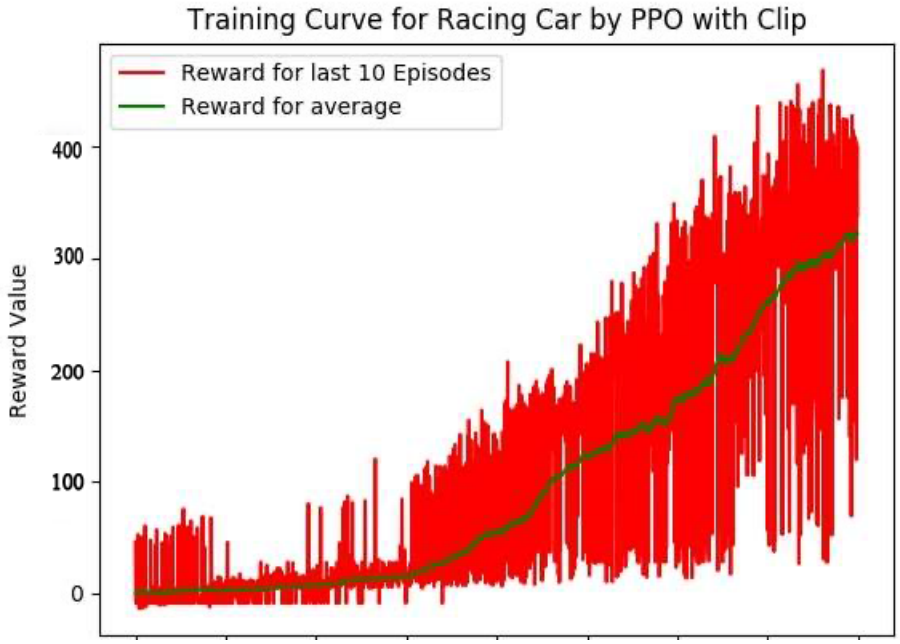

Fig. 10: Training Reward for PPO

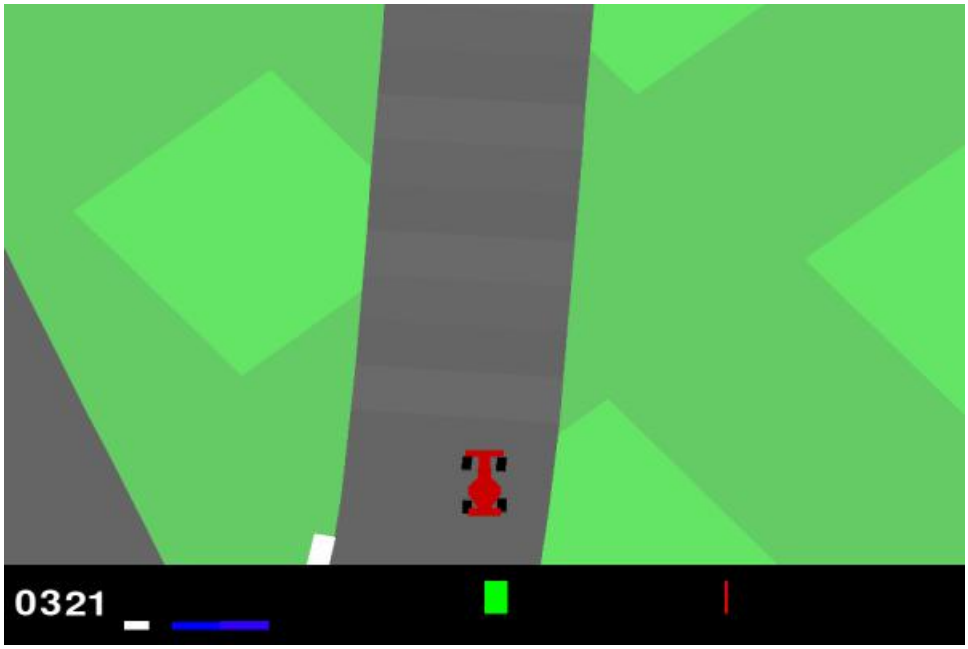

Fig. 11: Simulation Rendering for PPO during testing

REFERENCES

[1] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. doi:10.48550/ARXIV.1312.5602

[2] Zhang, Y. (2020). Deep reinforcement learning with mixed convolutional networks. arXiv preprint arXiv:2010.00717.

[3] Holubar, M. S., & Wiering, M. A. (2020). Continuous-action reinforcement learning for playing racing games: Comparing SPG to PPO. arXiv preprint arXiv:2001.05270.

[4] Hu, Z., & Kaneko, T. (2021, August). Hierarchical Advantage for Reinforcement Learning in Parameterized Action Space. In 2021 IEEE Conference on Games (CoG) (pp. 1-8). IEEE.

[5] Kakade, S., & Langford, J. (2002). Approximately optimal approximate reinforcement learning. In In Proc. 19th International Conference on Machine Learning.