

# rraftec: A Deep Reinforcement Learning agent able to play the game of Crafter

Rares FOLEA

Computer Science & Engineering Department  
Faculty Of Automatic Control And Computers  
University Politehnica Of Bucharest  
Bucharest, Romania  
rares.folea@stud.acs.upb.ro

Andrei VATAVU

Computer Science & Engineering Department  
Faculty Of Automatic Control And Computers  
University Politehnica Of Bucharest  
Bucharest, Romania  
andrei.vatavu@stud.acs.upb.ro

**Abstract**—rraftec: A Deep Reinforcement Learning agent able for the game of Crafter [1].

**Index Terms**—Deep Reinforcement Learning, Crafter, Agent, DQN, Double-DQN, pytorch

## CONTENTS

I	Introduction	1
II	Implementation of rraftec	1
II-A	Neural NetV1	1
II-B	Neural NetV2	2
III	Training of rraftec	2
IV	Evaluating results of rraftec against Crafter	2
V	Appendix A: 4-seed DDQN rraftec-agents trained 1KK steps	4
VI	Appendix B: 4-seed DQN rraftec-agents trained 1KK steps	4
VII	Appendix C: A random agent performance	6

## I. INTRODUCTION

**Crafter** features randomly generated 2D worlds where the player needs to forage for food and water, find shelter to sleep, defend against monsters, collect materials, and build tools. **Crafter** aims to be a fruitful benchmark for reinforcement learning [1].

In our implementation, we did a benchmark between the performance of a random agent and two Deep Reinforcement Learning agents, based using **DQN**(Deep Q-learning) and **Double-DQN**.

Because Q-learning evaluates the future maximum approximated action value using the same Q function as the current action selection policy, it can occasionally overestimate the action values in noisy situations, slowing learning. To address this, a variation known as Double

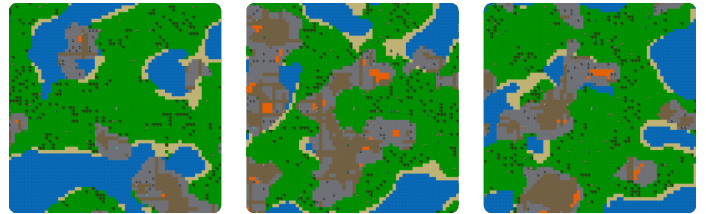


Figure 1: The possible maps of Crafter.

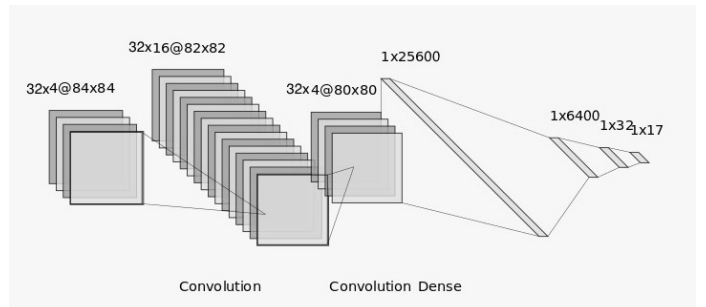


Figure 2: The Deep Convolutional NetworkV1 chose by us for the Deep Reinforcement Learning agent rraftec.

Q-learning has been developed. Double Q-learning[2] is an off-policy reinforcement learning system in which the value evaluation policy differs from the policy used to pick the next action.

## II. IMPLEMENTATION OF RRAFTEC

Our implementation for an agent able to play this game is available open-source, at:

<https://github.com/raresraf/rraftec>

We have tried two approaches for the neural network.

### A. Neural NetV1

The architecture for the deep neural network that we have used is:

Listing 1: Implementation of the network using pyTorch

```
def get_estimator(action_num, device,
                  input_ch=4,
                  lin_size=32):
    return nn.Sequential(
        nn.Conv2d(input_ch, 16, kernel_size=3),
        nn.ReLU(inplace=True),
        nn.Conv2d(16, 4, kernel_size=3),
        nn.ReLU(inplace=True),
        View(),
        nn.Linear(4 * 80 * 80, 80 * 80),
        nn.ReLU(inplace=True),
        nn.Linear(80 * 80, lin_size),
        nn.ReLU(inplace=True),
        nn.Linear(lin_size, action_num),
    ).to(device)
```

### B. Neural NetV2

We have also trained the model on a slightly different network, where we have seen a little improvement in terms of rewards.

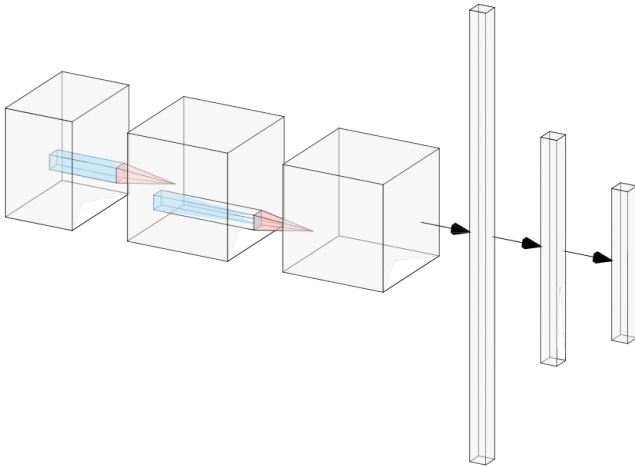


Figure 3: The Deep Convolutional NetworkV2 chose by us for the Deep Reinforcement Learning agent rraftec.

Listing 2: Implementation of the second network using pyTorch

```
def get_estimator(action_num, device,
                  input_ch=4,
                  lin_size=32):
    return nn.Sequential(
        nn.Conv2d(input_ch, 8, kernel_size=3),
        nn.ReLU(inplace=True),
        nn.Conv2d(8, 8, kernel_size=3),
        nn.ReLU(inplace=True),
        View(),
        nn.Linear(8 * 80 * 80, 8 * 80),
        nn.ReLU(inplace=True),
        nn.Linear(8 * 80, lin_size),
```

```
nn.ReLU(inplace=True),
nn.Linear(lin_size, action_num),
).to(device)
```

In our implementation, certain modules have been used from the previous labs of AAIT@UPB, containing implementations of the **DQN** and the **Double-DQN**.

Link to the resource:

<https://colab.research.google.com/drive/1B1sQXuyyTkfHza9Pw5kAUUaSEX2FWPmY>

### III. TRAINING OF RRAFTEC

In the training, we have used the following *params* and *hyperparams* for **DQN** and **DDQN**:

A replay memory:

(size=1000, batch\_size=32),

Adam optimizer for the network:

(learning rate=1e-3, eps=1e-4),

linear decay epsilon schedule:

(start=1.0, end=0.1, steps=100000),

Warmup steps: 10000,

Update steps: every 1 iteration.

We have trained the model using 4 seeds, each for **1000000** steps.

### IV. EVALUATING RESULTS OF RRAFTEC AGAINST CRAFTER

We have evaluated every 10000 the reward obtained by the rraftec agent against Crafter. The total training took **1000000** steps, and at the end of the training, **Double-DQN** (clearly) have overall performed better than the initial *random* agent on the game of Crafter. **DQN** showed only some slightly improvements from the initial *random* agent on the game of Crafter, even though there were episodes in which **DQN** still performed worse than the random agent.

**Double-DQN** has clearly outclassed **DQN** in terms of reward obtained.

For **Double-DQN**, it seems that the agent has constantly learned to perform the collection of wood and planting a plant, being rewarded for these two achievements: achievement\_collect\_wood and achievement\_place\_plant. Also, the **Double-DQN** agent is constantly receiving the achievement\_wake\_up reward.

Sometimes, the agent is able to even collect the saplings, receiving the achievement\_collect\_sapling reward. Rarely, the agent finds the lake and achieves the achievement\_collect\_drink reward. Very rare, the agent gets the achievement\_defeat\_zombie reward or the achievement\_eat\_cow.

### REFERENCES

- [1] Danijar Hafner. Benchmarking the spectrum of agent capabilities. *arXiv preprint arXiv:2109.06780*, 2021.

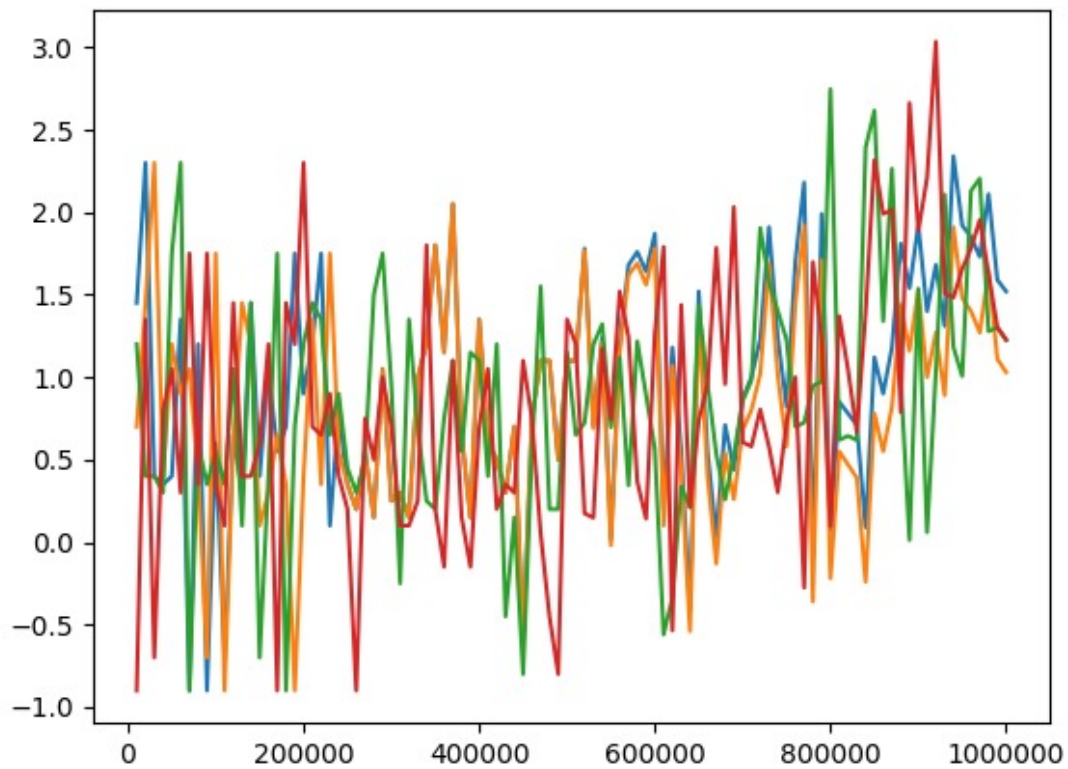


Figure 4: Training the Deep Reinforcement Learning agent rraftec using the DDQN strategy on the game of Crafter.

- [2] Hado Hasselt. Double q-learning. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010.

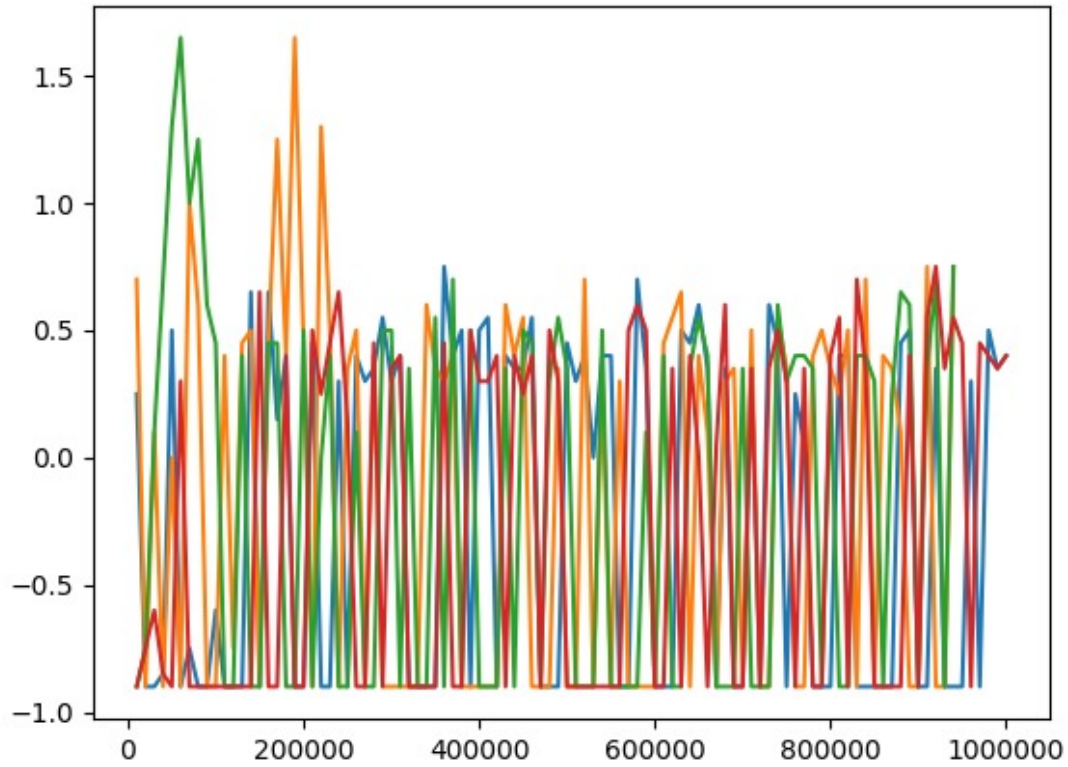


Figure 5: Training the Deep Reinforcement Learning agent rraftec using the DQN strategy did not show good performances on the game of Crafter.

#### V. APPENDIX A: 4-SEED DDQN RRAFTEC-AGENTS TRAINED 1KK STEPS

This appendix shows 4 training episodes of the Deep Reinforcement Learning agent rraftec using the **Double-DQN** strategy and the NeuralNetV2.

#### VI. APPENDIX B: 4-SEED DQN RRAFTEC-AGENTS TRAINED 1KK STEPS

This appendix shows 4 training episodes of the Deep Reinforcement Learning agent rraftec using the **DQN** strategy and the NeuralNetV2.

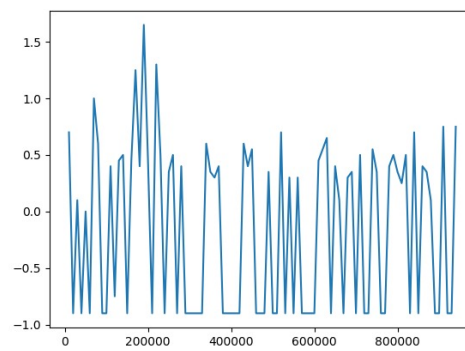
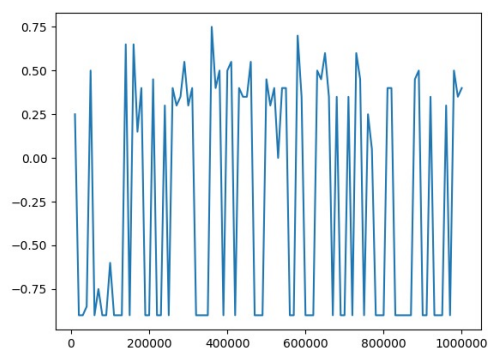
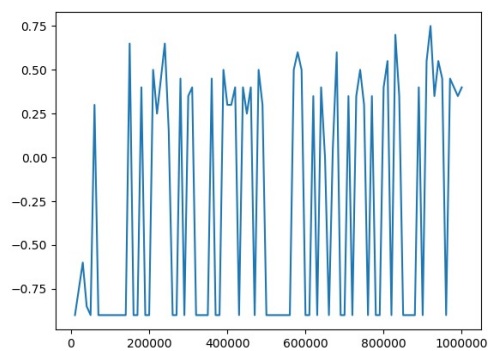
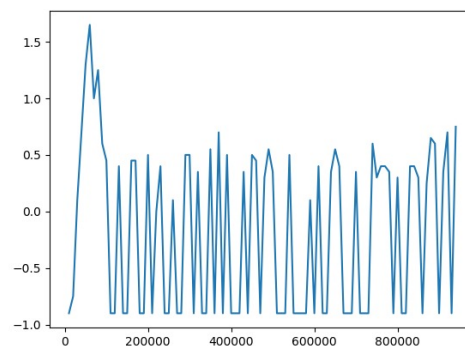
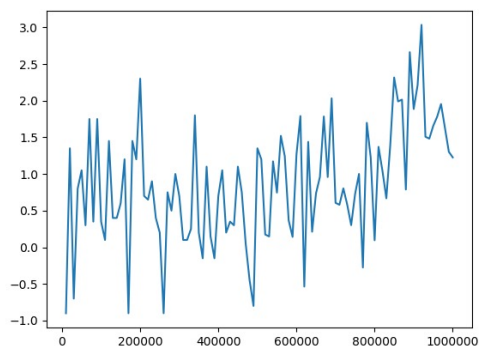
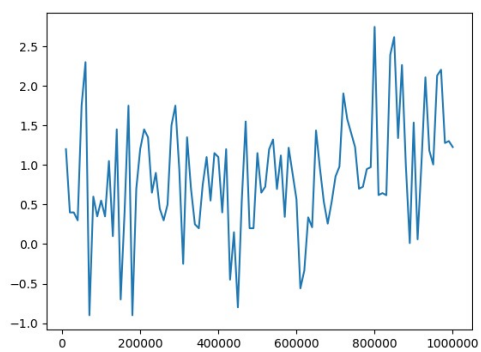
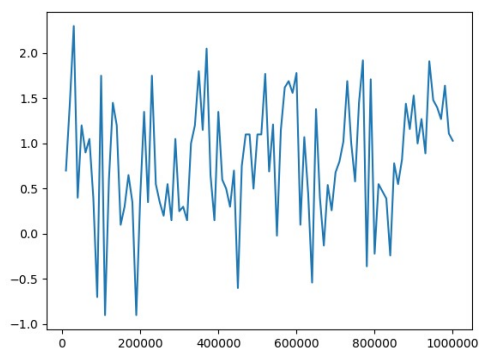
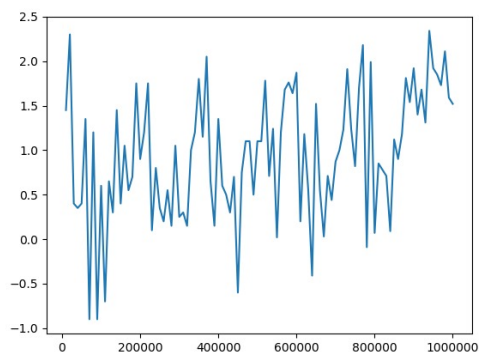


Figure 6: 4-seed training of the Deep Reinforcement Learning agent rraftec using the DDQN strategy.

Figure 7: 4-seed training of the Deep Reinforcement Learning agent rraftec using the DQN strategy.

## VII. APPENDIX C: A RANDOM AGENT PERFORMANCE

We have analyzed the performance on Crafter of a random agent.

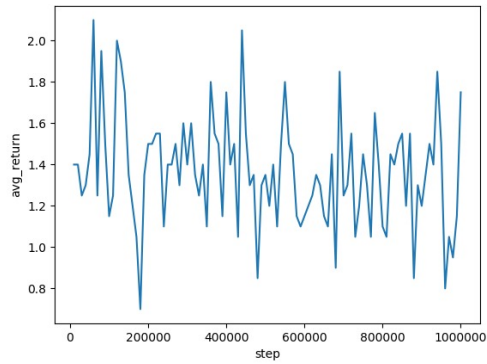


Figure 8: The performance of a random agent performance.