Reinforcement Learning- Final Course Project

Daniel Itkes (ID. 314538844),Tomer Juster (ID. 206765372)

Submitted as final project report for the RL course, IDC, 2023

1 Introduction

As part of the Reinforcement Learning course in Reichman University we have invested a semester learning the theoretical concepts of RL, and as a final assignment we were tasked to use these ideas and implement an agent that will be able to solve Sokoban, a 2D grid game in which the player is supposed to push boxes into specific targets on the map. In this report we will cover our attempts to train the agents, starting from the very first simple RL algorithm we have ever implemented and down to using existing RL frameworks and advanced visualization tools. During our work we have refined our understanding of the topics seen in class (and in Deep Learning, which we haven't experienced before), and feel we are in a good position to dwell more deeply into RL in the future.

1.1 Related Works

During our worked we have taken inspiration from the work presented in the following GitHub repositories: Sokoban-gym, RL-Experiments. Our original approach was to implement our own DQN algorithm, during our attempts we have refined our reward mechanism based on the first repository. Our second and final approach was to use the "stable baselines3" RL framework, and we have referenced the work in the second repository to set up our training process.

2 Solution

2.1 General approach

The original approach was to implement the algorithms from scratch. We have decided on the DQN algorithm which we have seen in class, as we were most familiar with it, which we thought was important for our first implementation of an RL algorithm. We have later switched to the "stable baseline3" framework upon realizing we don't have the necessary Deep Learning knowledge to optimize our network, and have experimented with the out-of-the-box A2C,PPO and DQN models the framework offers. After a couple of experiments we have decided on continuing with the PPO model, letting the framework do the heavy lifting and focusing on reward shaping.

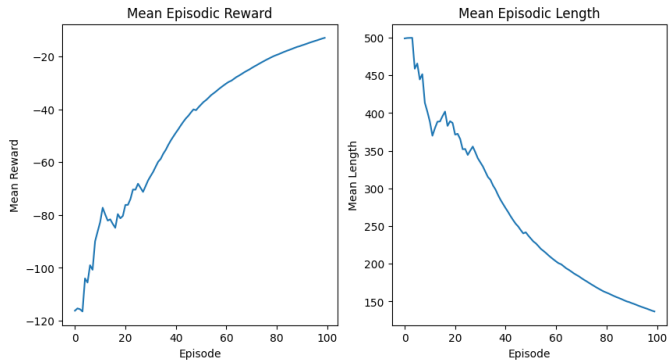


Figure 1: Ex1 Experiment Results

2.2 Design

Our main results came from models from the ”stable baseline3” framework. We have created a custom environment which wraps the original ”PushAndPullSokobanEnv” in order to get control over the interaction of the agent with the environment. Due to the limited amount of free resources in ”Google Colab”, we have opted to train the models locally using simple python script that creates/loads models as needed, we have connected the models to the ”TensorBoard” framework in order to keep track of the progress during training. The training time varied between the different models we have used, we have created and discarded numerous models throughout the project.

3 Experimental results

3.1 Ex1 - Fixed Scenario - One Box

In the first exercise we were tasked with training an agent which will solve a fixed environment. As this was our first real brush with implementing RL algorithms, we have opted to go back to the lecture notes and recordings and implement the DQN algorithm. Our model has handled the task relatively with easy, with our initial configurations and reward shaping mechanism being very basic. The reward shaping mechanism we have implemented used the initial environment rewards, along with slight penalty’s for not moving the box, staying in place and a general penalty every step to encourage faster convergence. Our model has converged after only 100 episodes, which amounted to less than 10 minutes. The agent has solved the environment optimally 100/100 times with an average episode length of 9 as demonstrated in Figure 1.

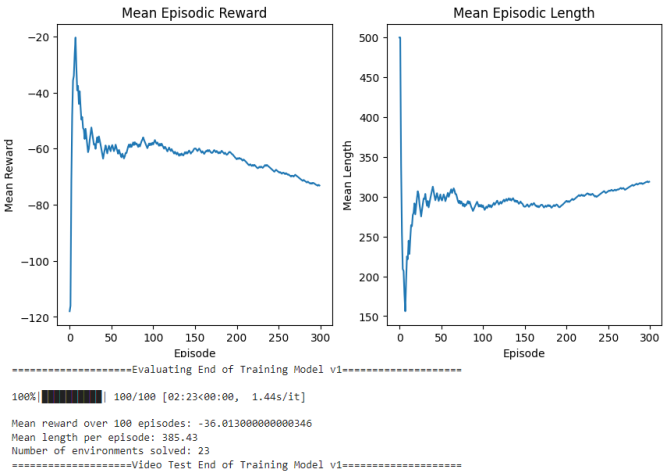


Figure 2: Experiment 1: DQN Version 1

3.2 Ex2 - Random Scenario - One Box

We can split the second exercise into two main parts. The first is a straight continuation of our attempts in Ex1, we have used the same settings as before and have later modified the reward mechanism. Neither have produced adequate results and by this point we have used up the available resources on the ”Google Colab” platform, and have opted to change directions. The second part was using the out-of-the-box models supplied by the ”stable baselines3” library, and working locally on our machines. In this subsection we will display the results of our different experiments.

3.2.1 Experiment 1: DQN Version 1

We have used the same settings as in Ex1, with a DQN model and a reward mechanism that is consisting of the original reward system along with additional penalties for not moving, not moving the box. Our results clearly show that our basic approach is inadequate, and that we should polish our reward mechanism as the agents performance was deteriorating in terms of the mean episode reward and mean episode length, as shown in Figure 2.

3.2.2 Experiment 2: DQN Version 2

The main difference in this experiment is the reward shaping mechanism. We have realized that in order to help our agent learn we need to carefully decide how and why we should penalize / reward it in order to not confuse it and only push it in the right direction. We have noticed in the video tests that the agent tends to roam aimlessly throughout the environment, and we concluded that we aid it to achieve the actual goal of the game, which is to push the box into the target cell. Our approach was to achieve two main things:

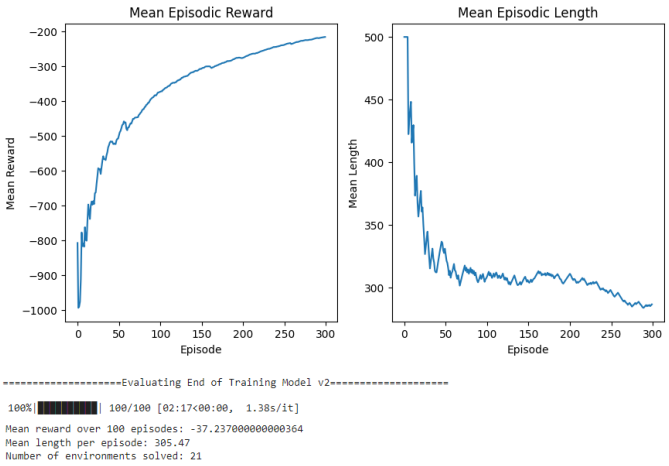


Figure 3: Experiment 2: DQN Version 2

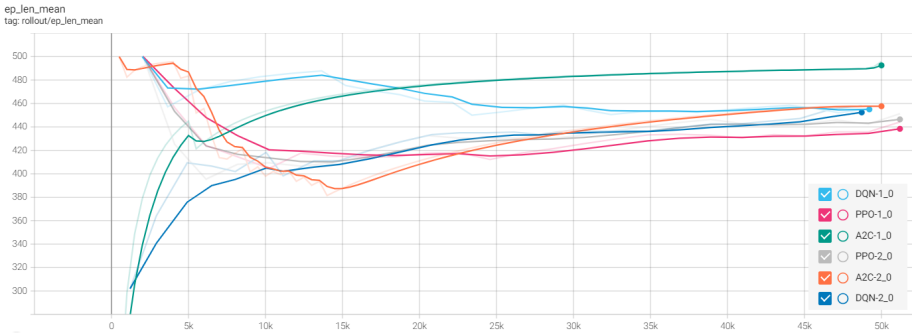
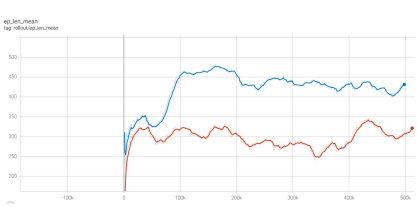


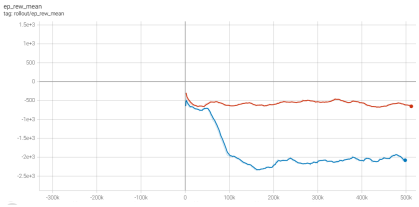
Figure 4: Experiment 3: Evaluating Different ”Stable Baselines3 Models

1. Get the agent to understand that it needs to move towards the box.
2. Get the agent to understand that it needs to push the box towards the target.

We have included a mechanism that penalizes / rewards the agent based on the euclidean distance between itself and the box, and between the box and the target, as the agent is getting closer to the box and if the agent pushes the box closer to the target it will be rewards, and vice versa. We have added an additional penalty for not moving at all. The results showed a clear improvement over time, as shown in Figure 3, but at this point, considering other students experiences, we have opted to ditch the self implementation approach and move to ”stable baselines3”.



(a) Mean Episode Length



(b) Mean Episode Reward

```
--------------------Evaluating End of Training Model DQN-v3-1--------------------

100%|██████████| 100/100 [02:31<00:00,  1.52s/it]
Mean reward over 100 episodes: -47.48700000000037
Mean length per episode: 480.37
Number of environments solved: 5
--------------------Evaluating End of Training Model PPO-v3-1--------------------

100%|██████████| 100/100 [02:04<00:00,  1.25s/it]
Mean reward over 100 episodes: -29.063000000000287
Mean length per episode: 335.73
Number of environments solved: 41
```

(c) Success Rate

Figure 5: Experiment 4: PPO/DQN Without Reward Shaping

3.2.3 Experiment 3: Evaluating Different ”Stable Baselines3 Models

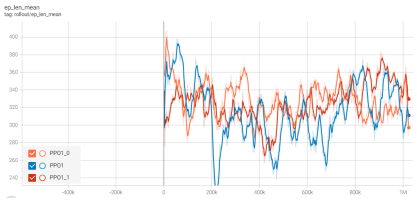
In this experiment we wanted to see the performance of different models supplied by ”stable baselines3” on the task at end. We have made a wrapper environment for the ”PushAndPullSokobobanEnv” in order to be able to modify the reward mechanism, but have not made significant changes from the original environment in this experiment. We have ran 2 instances of each of the following models:

1. DQN
2. A2C
3. PPO

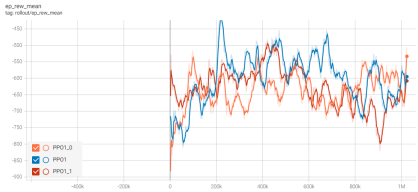
It was at this point in the project we have understood the vast randomness in RL algorithms. We have realized that the performance of the agent is highly dependant on its random seed, and that two identical models with the same settings can have different results. The purpose of this experiment was to pick an algorithm that we will focus on, we have ran several such experiments and have determined that we should use the PPO and DQN algorithms with Multi-Layer Perceptron Policy, as it gave better results as shown in Figure 4, and because it was the general advice from students who have worked with ”stable baselines3” in this project.

3.2.4 Experiment 4: PPO/DQN Without Reward Shaping

After our experimentation with the framework we have decided to let the models train for some time. Our results show that the PPO model has achieved better performance on both Mean Episode Length and Mean Episode Reward metrics as shown in Figure 5. Furthermore our evaluation shown in Figure 5 clearly demonstrates PPO (at least with the current initialization) is performing better after 500K timestamps. Based on our results we have opted to focus only on the PPO model.



(a) Mean Episode Length



(b) Mean Episode Reward

```
--------------------Evaluating End of Training Model PPO1_0 with Env1-----

100%|██████████| 100/100 [02:01<00:00,  1.21s/it]

Mean reward over 100 episodes: -43.66700000000028
Mean length per episode: 334.77
Number of environments solved: 38

--------------------Evaluating End of Training Model PPO1 with Env2-------

100%|██████████| 100/100 [01:42<00:00,  1.03s/it]
Mean reward over 100 episodes: -222.19499999999877
Mean length per episode: 284.79
Number of environments solved: 48

--------------------Evaluating End of Training Model PPO1_1 with Env3-----

100%|██████████| 100/100 [01:58<00:00,  1.19s/it]
Mean reward over 100 episodes: -197.44399999999995
Mean length per episode: 320.34
Number of environments solved: 39
```

(c) Success Rate

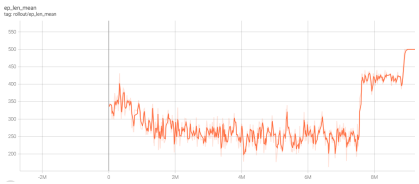
Figure 6: Experiment 5: PPO with different Reward Shapings

3.2.5 Experiment 5: PPO with different Reward Shapings

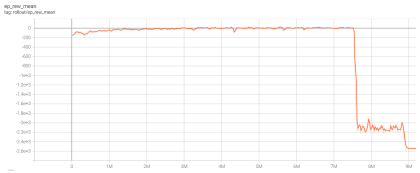
Based on our results in previous experiments, we have decided to add the reward mechanism we have used in experiment 2 with a few modifications. Recall that our reward mechanism in every given time step is as follows:

1. +/- Big reward based on distance of box from target.
2. +/- Small reward based on distance of box from agent.
3. - Small reward based on unchanged room state.

During our previous experimentation we have seen a tendency of the agent to move from side to side, we had a few hypothesis is why it was the case. First, in our reward for distance from box to agent we have stopped rewarding the agent if he is adjacent to the box, and we penalized it harshly if he pushed the box in the wrong direction. This may have lead the agent to be just whit in reach of the box but without actually moving it, even though the reward for pushing the box is large as well. Another possibility is that the agent felt comfortable moving in and out of a cell adjacent to the box, as it received the same penalty and reward for moving closer/further to the box with the rewards canceling out. We have attempted to address these issues by defining three new environments, each with its unique modification to our reward system from experiment 1: The first environment will penalize/reward the agent based on whether it pushed the box if it is adjacent to the it. The second environment will penalize/reward the agent for every step to encourage faster convergence. The third environment will penalize/reward the agent if the cell it is in was already visited in the last X steps (we choose x = 4). We will define these environments as Env1,Env2,Env3 and the results are shown in that



(a) Mean Episode Length



(b) Mean Episode Reward

```
====================Evaluating Middle of Training Model PPO2_0=====

100%|██████████| 100/100 [01:57<00:00,  1.17s/it]
Mean reward over 100 episodes: -231.20899999999972
Mean length per episode: 246.08
Number of environments solved: 55
====================Evaluating End of Training Model PPO2_0========

100%|██████████| 100/100 [01:45<00:00,  1.06s/it]
Mean reward over 100 episodes: -244.89499999999992
Mean length per episode: 249.53
Number of environments solved: 53
```

(c) Success Rate

Figure 7: Experiment 6: PPO CNN Policy

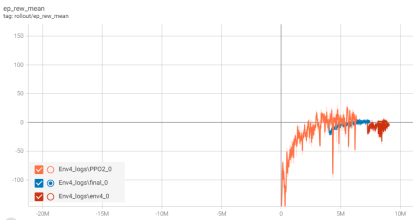
order in Figure 6. We can conclude a few things from the results. First, it seems that in terms of achievements, the agents in all environments are performing similarly, with the agent in Env2, which penalizes reward in every time steps, performing better. More concerning is the fact that the models don't show clear improvements. It seems playing with our existing reward mechanism has its limitations, and better approach is needed. We can up with two main options:

1. Play around with the hyper parameters.
2. Switch the policy in the models definition.

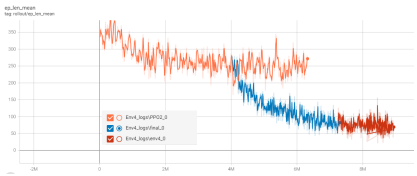
As we have not experienced tuning hyper parameters before in an experimental setting, and as we felt that we don't know enough about the implementations of the models in the framework to experiment with such a thing, we have opted to continue with option 2.

3.2.6 Experiment 6: PPO CNN Policy

In this experiment we have used PPO with CNN Policy and taken the same reward mechanism as in experiment 2, and have let the model train for some time. As shown in Figure 7 the model has been in and around 250 Mean Episode Length and +/-0 Mean Episode Reward for the majority of its training. After a few video tests we have seen that the model tends to stay in the spot without moving, which prompted us to try and increase the penalty for every step in which the room state has not changed. This has lead to the sharp drop in the models performance, and when we opted to try and change the environment back to how it was it finally killed off the training process and the model was stuck on the worst possible performance.



(a) Mean Episode Reward



(b) Mean Episode Length

```
====================Evaluating End of Training Model env4====================
100%|██████████| 100/100 [00:14<00:00,  6.77it/s]
Mean reward over 100 episodes: 6.941499999999974
Mean length per episode: 33.01
Number of environments solved: 95
```

(c) Success Rate

Figure 8: Ex3 Experiment Results

```
====================Evaluating End of Training Model env5 on Ex2====================
100%|██████████| 100/100 [00:39<00:00,  2.55it/s]
Mean reward over 100 episodes: -15.17249999999982
Mean length per episode: 95.65
Number of environments solved: 84
```

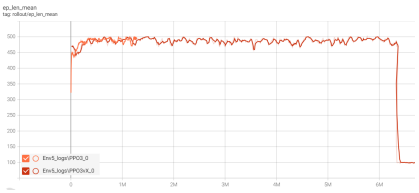
Figure 9: Experiment 8: Using Ex3 Model For Ex2

3.2.7 Experiment 7: PPO No Reward Shaping

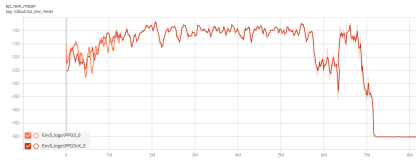
In this experiment we have taken our best performing model from experience 6 and have removed all the reward shaping mechanism. The rational is that as the agent has converged in the previous experiment it has a basic understanding of the game, and we should release it to try and play in the original ”PushAndPullSokobanEnv”. This has produced good results and the agent has reached around 100 Mean Episode Length and has stayed there for quiet some time. We have then made a few video tests to check how the agent is behaving and have noticed that some times it tends to stay in place, which has prompted us to re-introduce a small penalty for each step that the game state has not changed. This has further increased the performance as shown in Figure 8. This is our best performing model in Ex2, and we believe with more training and tweaking we can make it even better. The graphs show how we take our model from experiment 6 and make the changes mentioned. Note that the model has not converged and we could have let it train further. We believe we are very close to ”solving” Ex2 but due to a lack of time were not able to.

3.2.8 Experiment 8: Using Ex3 Model For Ex2

We were curious to see how our best performing model in Ex3 was performing in Ex3. We did not ”fine tune” the model back on Ex2 for lack of time but the logic was that the model should perform better in a simpler task. The results ’were not that surprising and are pretty good as seen in Figure 9, and it is worth noting that the model has taken an interesting journey along the way, training on Ex2 in different ways, then training on



(a) Mean Episode Length



(b) Mean Episode Reward

```
--------------------Evaluating Middle of Training Model PPO3vX_0-------------

100%|██████████| 100/100 [03:22<00:00,  2.03s/it]
Mean reward over 100 episodes: -100.36000000000087
Mean length per episode: 500.0
Number of environments solved: 0

--------------------Evaluating End of Training Model PPO3vX_0---------------

100%|██████████| 100/100 [03:21<00:00,  2.02s/it]
Mean reward over 100 episodes: -100.23000000000087
Mean length per episode: 500.0
Number of environments solved: 0
```

(c) Success Rate

Figure 10: Ex3 Experiment Results

Ex3 and finally going back to Ex2, this was one of the concepts we saw in class and it was great to see it in practice.

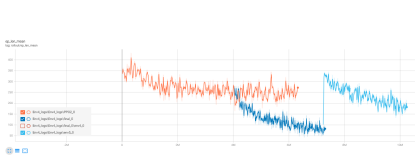
3.3 Ex3 - Random Scenario - Two Boxes

3.3.1 Experiment 1: PPO With Reward Shaping

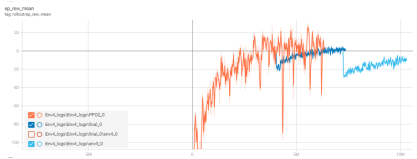
This task was extremely difficult, and in retrospect it seems that we haven't approached in the right direction. Our idea was to use the same concept of a reward system to train a PPO model with CNN policy, and just leave it to train in the hope it will eventually learn enough to get going. We have seen implementations on the Internet that used an enormous number of episodes with little to no success at the first thousands of episodes, and thought it just takes the model some time to figure things out. This turned out not to be the case, it seems that the model just refused to move and always stayed in its place no matter what we have tried in terms of rewards. In this experiment the model eventually converged in terms of rewards, but still stayed in its place most of the time, which lead us to incrementally add to the penalty for the room state staying the same, which had no results whatsoever. Eventually we have tried to play around with the maximum steps allowed for a model to solve an environment to try and get it to move around more but that has killed the model performance and we were forced to drop it, as our approach did not work. Figure 8 shows our attempts at training the models, using incremental changes to the penalty for not moving. The graphs show two models, the first model is not using custom reward shaping at all, while the second uses the same approach as previous experiments.

3.3.2 Experiment 2: PPO With Best Performing Model From Ex2

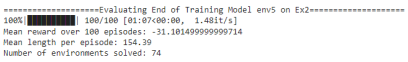
We have decided to let our best performing agent from Ex2 to train on Ex3. This approached greatly improved our results as can be seen in Figure 11. We have not



(a) Mean Episode Length



(b) Mean Episode Reward



(c) Success Rate

Figure 11: Experiment 2: PPO With Best Performing Model From Ex2

used any reward shaping at all, for the same reason as in experiment 7, where we hypothesized that "unleashing" a model after a period of heavy reward/penalties makes it perform better. Note that the model hasn't converged and we anticipate it to keep improving over time. We think the strategy of training a model until convergence and only then experimenting with different reward mechanism is effective. This is our best performing model for Ex3. In Figure 11 you can see how we took the different retrained models in order to refine our training strategy.

3.3.3 Ex3 - End Notes

If we were to start over and come back to this problem in the future, we think a good way to approach this problem will be to let it incrementally solve more and more fixed environments so the agent will get a sense of the task. After the agent will have a good amount of environment which it has converged on, only then we would proceed to train it regularly and monitor the performance to decide if we should remain in first step for a while longer or it was train normally. This is of course take an extraordinary amount of time and resources, but we feel it would have been a good approach.

4 Algorithms used and comparison

In the first part of the assignment we have implemented the DQN algorithm, which wasn't straight forward especially as we did not have any background in Deep Learning. The challenging part of to familiarize ourselves with the technical aspect of working with PyTorch and tensors, which wasn't straight forward. The main advantage is of course that we were familiar with the algorithm from class, and had fun implementing it ourselves. In the second part we have used PPO which is an algorithm that optimizes the control of a system in a way that is more stable and efficient compared to other policy gradient methods. It works by updating the policy so that the change is not too big from one step to the next, avoiding issues that can arise from large policy updates. The main advantage is that the model performed very well and came out of the box, but as such we didn't feel comfortable with tweaking with its hyper parameters and it has limited us.

5 Discussion

In the last two weeks we have familiarized ourselves with the concepts of neural networks and how to use them to implement the algorithms seen in class. However, our initial success with the fixed environment gave us confidence in our ability to tackle the more challenging random scenario. Unfortunately, the basic approach with DQN proved not enough, and the need for a more sophisticated reward mechanism became apparent.

In the course of our project, we transitioned to the "stable baselines3" framework, experimenting with various algorithms, and ultimately deciding on the PPO model as the most promising approach. This decision was based on the observations we made and the evaluation of the performance, efficiency, and adaptability of the model over time.

Our exploration into the reward shaping was particularly enlightening but also filled with challenges. The rewards and penalties had to be finely balanced to direct the agent's behavior without confusing it. Though we saw some improvements with specific tweaks to the reward system, the limitation of our chosen approach became clear as we grappled with the random behavior of the agent and stagnation in learning.

Experimentation with different policies and the frustrating experience with a CNN policy further highlighted the complexity and sensitivity of RL models. Small changes could either lead to minor improvements or render the model completely ineffective.

The final part of the project, where we struggled to get the model to move and adjust to the dynamic environment, underscored the importance of a more systematic approach and the consideration of incremental learning strategies. If we were to redo this project, we would explore letting the agent incrementally solve more straightforward environments to build a foundational understanding of the task.

In conclusion, our semester-long journey into RL allowed us to not only deepen our understanding of the theoretical concepts but also get our hands "dirty" with implementing the algorithms we have learned. We learned the importance of reward shaping, the challenges of training an RL agent, and the nuances of different RL frameworks and models. While the results were not always as successful as we hoped, the insights gained from this hands-on experience will undoubtedly be valuable as we continue to explore and engage with the field of Reinforcement Learning in the future.

6 Results

1. Ex1: 100/100 Successful runs, ELM: 9 (Figure 1)
2. Ex2: +/- 95/100 Successful runs, ELM: +/- 33 (Figure 8)
3. Ex3: +/- 75/100 Successful runs, ELM: +/- 74 (Figure 11)

7 Code

Colab Notebook, GitHub Repository