Reinforcement Learning Project with the MiniHack Environment

Xiaoyan HONG, Xincheng LUO, and Joshua FAN

ESSEC & CentraleSupélec

1 Introduction and motivation

This project is a deep dive into the field of reinforcement learning, and a thorough exploration of how agents and environments interact with each other. We first started by creating a customized MiniHack game that consists of 3 different levels. Each level has a different environment for us to develop and test the RL algorithms. We then deploy agents of different algorithms if they can learn how to play the game well within limited learning epochs. The aim for this project is to learn how to design the environment and how to deploy the agent to interact with the environment, at the same time, compare the performance among different algorithms.

2 Description

2.1 Description of the environment

The designed MiniHack game which is about navigation task has three different levels with increasing complexity, each with a different set of obstacles and respective penalties. The idea is to make observations on how agents will react to changing environments.

1. The environment in the primary level only has an agent, trees and walls, and the agent should somehow find the fixed exit through learning while avoid running into an obstacle;

2. The second level has an agent, obstacles, a sink, a teleport portal and a moving monster. If the agent comes across the monster or drops into a sink, it will lose points or even die;

3. The third level includes an agent, obstacles, 2 moving monster and 2 portals which transfers the agent to a closer location to the exit, 2 sinks, an axe and an apple. In addition, the exit in this level is randomly intialized for each play.

Besides the default reward for reaching the target as defined in the original environment, we designed a customized reward function to convert the sparse reward distribution into a much denser one, so that the agent can learn how to approach the exit quickly. The intuition behind is that, the closer the agent is to the exit, the larger the reward for each step. And extra bonuses for reaching

certain distances, so that the agent is less easily to traverse back once it gets close enough to the exit.

In the latter two advanced levels, it is worth mentioning how the features are interacting with the agent. In the second level of the environment, there are some unique features. The target has a fixed position at the bottom right corner of the grid map. A monster that represents a terminal of the agent is placed randomly on the grid map as well as a teleport gate and a sink, with different penalties assigned respectively. Additionally, custom obstacles in the form of terrain blocks are placed at specific locations on the grid.

In the third level of the environment, there are some unique features as well. Aside from leveling up the complexity of environment by adding one more monster, one more sink, and one more portal, there are extra negative-rewarded items placed randomly in the grid.

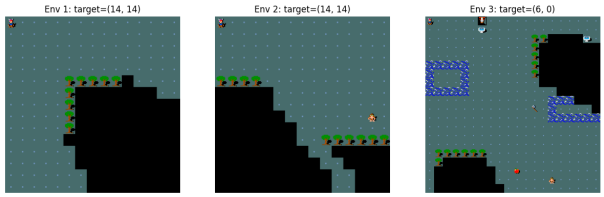
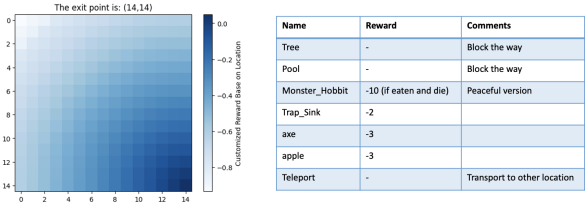


Fig. 1. An Overview of the Environment



Name	Reward	Comments
Tree	-	Block the way
Pool	-	Block the way
Monster_Hobbit	-10 (if eaten and die)	Peaceful version
Trap_Sink	-2	
axe	-3	
apple	-3	
Teleport	-	Transport to other location

Fig. 2. The Reward Map (Location & Object)

An action space of 8 different actions is also included in the environment. Here is how we map the actions to the navigation position that the agent takes – 0: up, 1: right, 2: down, 3: left, 4: up right, 5: down right, 6: down left, 7: up left.

2.2 Description of the observation space

Before going into the specifics of the agents, it is necessary to understand what the agents are able to see or observe while learning in the environment, that is,

what is the given observation to be based on for further actions. In this project, we used mainly two different types of observation keys that are provided from MiniHack and NLE, tty_cursor and chars_crop.

The observation key tty_cursor is defined as the position of the terminal cursor in the environment or game. In the context of what we created, the indicator here is referring to the agent's coordinates on the game screen. This is essential because what we would like the agent to observe is not partial information of the environment but a clear view of what is happening in the game so that it can based on the observation to take specific actions to navigate to the target position. However, it is important to note that the usefulness of the observation key depends on the purpose of the game and the type of information required to perform well. In some cases, such as when negative rewarded obstacles exist, tty_cursor alone may not provide enough information for the agent to make the best decisions.

We also used chars_crop as an observation key, which provides a cropped area of the grid screen, restricting the agent's vision to its immediate surroundings. This allows for better decision-making based on relevant information and simulates real-life situations where individuals have a limited view.

We also tried using "glyphs" as the observation key for CNN-DQN, which represents the entire game in 5992 unique entities. This allows the agent to easily understand the game state and make informed decisions since we formatted it to be neural network-readable.

2.3 Description of the agents

The first agent we implemented is a q-learning agent that determines the best action given an observed state. The agent learns to navigate by exploring and exploiting the learned q-values, using an epsilon-greedy strategy to avoid local optimal result. The q-learning method involves acquiring an action-value function that maps a key-value dictionary, where the key is a pair of state and action, and the value is the corresponding q-value. We used the empirical hyperparameters, which were explored in previous assignments, for alpha(learning rate), gamma(discount rate), epsilon and ran the model for 200 iterations.

The second agent we implemented is an agent using the expected-SARSA algorithm, which is similar to q-learning but updates q-values differently. This algorithm calculates the expected q-value of the next state using the average q-values of all actions. We used the tty_cursor as an observation key which is the same as the q-learning agent.

Another agent that we tried is the DQN algorithm, that is, combining q learning and deep neural networks to learn the action value function that performs the best. In this agent implementation, we used a different observing key "chars_crop" to better gather information about the layout of the surrounding environment and obstacle positions. Although we managed to customize the agent to our environment, it was difficult for the agent to converge and reach the target within acceptable time duration. This is probably because the "chars_crop" provided by minihack has too many observation states, and we also

tried large or small tiles and CNNs while still failed to converge the algorithm. If we have time, we may delve into how to provide observation states suitable for dqn algorithm to learn.

3 Discussion and visualization of the results

In the experiment with the q-learning agent, we can see that the sum of rewards converged in three different levels. From the graph, we can see that the agent is learning overtime. For example in level 1, it first started with a reward of almost minus 500 per episode, then at the end it converged to less than minus 50 per episode. We also noticed the agent got lower sum of rewards for higher level at the beginning of training, which indicates that the agent spent more efforts to explore and find the exit when the environment grew more complex. The learning performance did not increase monotonically as the agent was carrying out exploration and exploitation at the same time, leading to a fluctuate graphs as shown below. It is also interesting to note that the agent converged faster in level 3, the main reason may be the target in level 3 is randomized and closer to the start position compared to default bottom left.

For expected-sarsa agent, it is noticeable that it converged slower than the Q-Learning agent and has higher variance as well. Since in expected-sarsa, the average q value of all actions in the next state is used, considering exploration as well as exploitation according to the epsilon greedy policy, resulting in a more conservative approach which can also be observed from the live play of both agents.

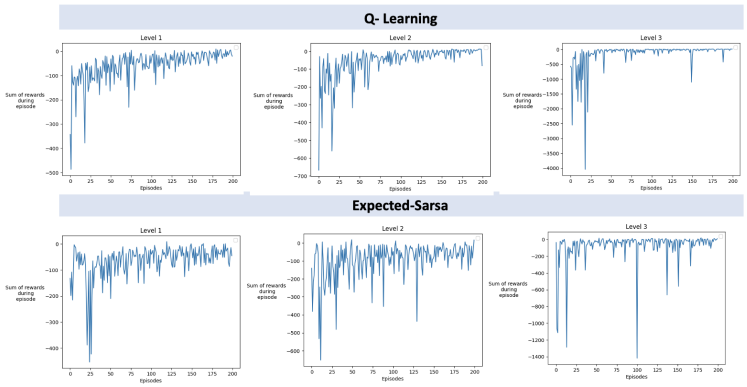


Fig. 3. An Overview of Performance with Both Agents

We also visualised the q value table learnt by the agent. From the graph below, it is shown that for each state action pair, how the q value distributes. The darker the blue shades are, the higher the q value is. This information can

be used to assess the performance of the q learning algorithm and fine-tune the agent's parameters.

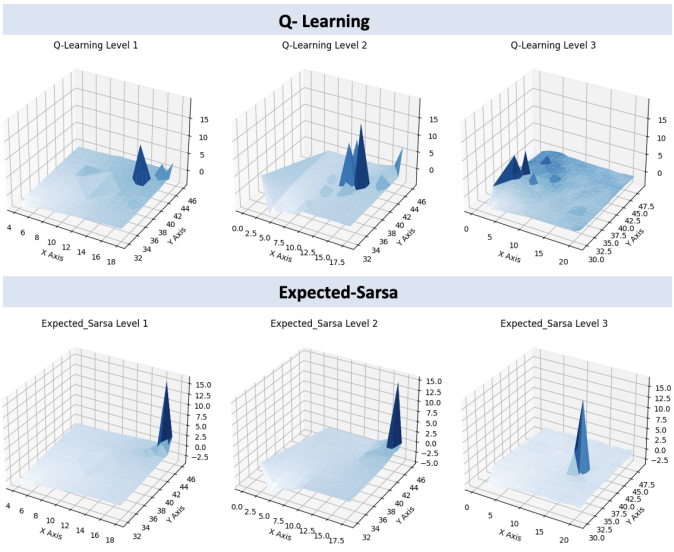


Fig. 4. Plots of Q value Tables in Each Level for Both Agents

From the comparison of the state value functions from the q-learning agent and the expected-sarsa agent, we can see that a more pronounced, concentrated, and taller (larger value on the z-axis) dark blue shade is shown in the latter plot. This indicates that the expected-sarsa agent has found a more certain and potentially more optimal path in the goal area of the state space than the q-learning agent. In the contrary, the q-learning state value function seems bumpier which conforms to its radical learning approach.

3.1 Further Discussion

Here are some notes on what we can further do for this project.

1. Refining the reward function: The reward function we defined now is direct and rough. We can improve it by accurately reflecting the desired behaviour for the agent. For example, it may require the agent to explore and walk away from the target, avoiding monsters or obstacles, to reach the final exit.

2. Adding a memory component: Such as a recurrent neural network or Long Short-Term Memory network, can improve the agent's decision-making process in MiniHack by allowing it to better understand the environment's dynamics from past observations. This can mitigate the limitations of the current observation-based approach and lead to more informed decisions based on previous experiences.

References

1] Samvelyan, M., et al. (2021). MiniHack the Planet: A Sandbox for Open-Ended Reinforcement Learning Research. In Thirty-fifth Conf. on Neural Info. Proc. Sys. Datasets and Benchmarks Track (Round 1). Retrieved from https://openreview.net/forum?id=skFwlyefkWJ.
[2] the-beee, RL-algorithms: https://github.com/the-beee/RL-algorithms.
[3] kavilan-nair, deep-rl-minihack-the-planet: https://github.com/BrentonBudler/deep-rl-minihack-the-planet.