Reinforcement Learning

COMS4047A/COMS7053A

Assignment

MiniHack the Planet

Geraud Nangue Tasse (geraudnt@gmail.com)
Tamlin Love (tamlin.love1@students.wits.ac.za)
Shahil Mawjee (shahil.mawjee@students.wits.ac.za)

MiniHack[1] is a powerful sandbox framework for easily designing novel RL environments ranging from small rooms to complex, procedurally generated worlds (Figure 1). By leveraging the full set of entities and environment dynamics from *NetHack* [2],[1] one of the richest grid-based video games, *MiniHack* allows designing custom RL testbeds that are fast and convenient to use.

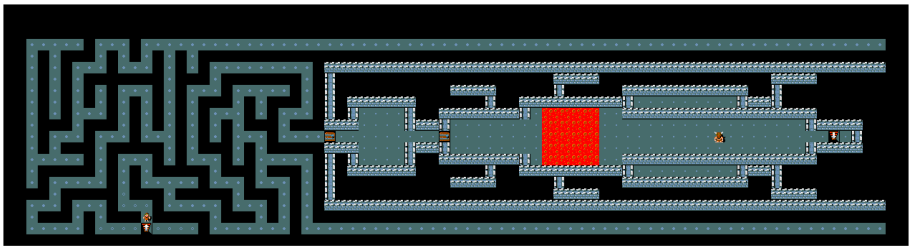


Figure 1: Quest-Hard: A complex task in *MiniHack* requiring multiple skills.

Your task is to create an agent that can navigate the *Quest-Hard* dungeon and accrue as many points as possible. Work in **groups of three to four people**. You must implement methods from **two of the following categories** to solve the problem.

- A value function method (e.g. DQN [3])
- A model-based method (e.g. MCTS [4])
- A policy method (e.g. REINFORCE [5])

[1] *NetHack* is a role-playing video game (RPG) released in 1987, in which the player must navigate through a procedurally generated dungeon while surviving its many hazards.

- A hierarchical method (e.g. Option-Critic [6])

If you wish, you may work in groups larger than four, but for every group member above four, you are expected to implement another two methods.

Within your selected methods, you may implement any kind of pretraining or modifications to the algorithm to train your agent. You will be submitting a report in which you describe your approach in tackling this problem and give an analysis of your agents' performances. **For any method/code you use that you find online, you must cite it properly, make at least some modifications (e.g augmenting the input, using subtask curricula, reward shaping, etc), and describe it in detail in the report.** Your report must include the following.

- The names and student numbers of each group member
- A full description of each algorithm used. This is important to show that you understand what you are doing, and cannot be directly taken from other sources
- A description of all the design decisions made when implementing each algorithm, and the motivations for these decisions. What worked? What didn't work?
- A list of hyperparameters used for each algorithm.
- A comparison plot of each agent, showing the reward per episode (averaged over several runs) obtained by each agent during training, in order to compare their rates of learning.
- A plot of how well you achieve sub-goals as a function of episodes (each sub-goal in a different colour), and one such plot for each agent. Examples of sub-goals are:
  - Reaching the maze exit
  - Crossing the lava river
  - Defeating the demon
  - Reaching the level's staircase
- A link to a video of at least one of your best runs of each algorithm, and a description of these.

We will be awarding marks for clever ideas, good descriptions and solid justifications.

In addition to the trained agent and the report, you will also be submitting your source code and a recording of your agents playing the game. Marks will be awarded for neat and reproducible code. A GitHub repository with a concise readme and well-commented code can go a long way towards getting these marks. Make sure your report contains a link to your GitHub repository.

1 Environment

The *MiniHack* package can be found on the GitHub repository https://github.com/facebookresearch/minihack, along with detailed installation instructions and a brief usage example. It is essentially a wrapper over the *NetHack Learning Environment (nle)* [2] that adds a number of tools to make it easy for creating custom environments. If you run into *nle* dependency errors, make sure to first install the *nle* package https://github.com/facebookresearch/nle.

The *MiniHack* package comes with a number of environments (described in the paper) [1], such as the *MiniHack-MazeWalk-15x15-v0* environment which rewards reaching the staircase down to the next level of the dungeon. You may use these subtasks, your own custom tasks, and/or reward shaping to train your agents if you wish, but **you will be evaluated using the *MiniHack-Quest-Hard-v0* environment**.

For more information on the environment, including the state and action space, refer to the documentation.

2 Video Submission

To visualise your agents' performance, you can use the nle dashboard application. Every time you run the *MiniHack* environment (created with $gym.make(env\_name, savedir = savedir)$), it automatically generates a csv file containing information on each episode, and a ttyrec file for each episode (located inside a zip file). First install the dashboard by following the instructions on the dashboard readme. Then navigate to the *savedir* directory generated by *nle* and locate your episode's csv and ttyrec files. Rename the csv to *stats.csv* and zip the ttyrec file(s) into *stats.zip*. Move *stats.csv* and *stats.zip* into the same directory. After starting the server and navigating to it through the browser, you can point the dashboard to this directory. If all goes well, you will be able to see your agent's run.

3 Evaluation

We will evaluate your agents on a number of runs and average the total reward. We will test your agents on unknown seeds, so be sure your agents are able to generalise and not overfit to the seeds you have trained on.

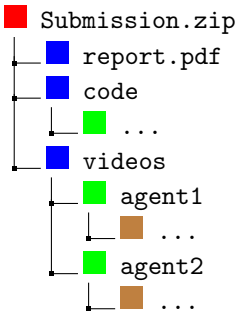
Your final mark will be comprised of the following.

- Your report
  - The detailed descriptions and motivations of your chosen methods
  - The detailed descriptions and motivations of your design decisions
  - The list of hyperparameters
  - The comparison plots

- Your source code
  - Marks will be awarded for neat, well-commented and reproducible code
- Your videos

4 Submission

Zip your report, video files and source code and upload it to the relevant Moodle submission page. For each agent, create a separate directory containing the best run video(s). For example, your submission may have the following directory structure.



References

[1] Mikayel Samvelyan, Robert Kirk, Vitaly Kurin, Jack Parker-Holder, Minqi Jiang, Eric Hambro, Fabio Petroni, Heinrich Kuttler, Edward Grefenstette, and Tim Rocktäschel. Minihack the planet: A sandbox for open-ended reinforcement learning research. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

[2] Heinrich Küttler, Nantas Nardelli, Alexander H. Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment, 2020.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[4] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

[5] R Williams. A class of gradient-estimation algorithms for reinforcement learning in neural networks. In *Proceedings of the International Conference on Neural Networks*, pages II–601, 1987.

[6] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.