Reinforcement Learning

COMS4047A/COMS7053A

Assignment

NetHack Learning Environment

Tamlin Love (tamlin.love1@students.wits.ac.za)
Shahil Mawjee (shahil.mawjee@students.wits.ac.za)

NetHack is a role-playing video game (RPG) originally released in 1987, in which the player character must recover the *Amulet of Yendor*, descending through a vast procedurally generated dungeon and surviving its many hazards to do so. It is considered among the hardest and most punishing RPGs in existence, which makes it a ripe challenge for reinforcement learning.

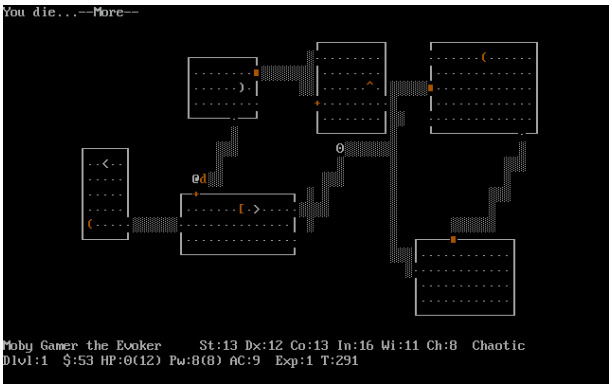


Figure 1: An example of *NetHack's* ASCII interface

Your task is to create an agent that can navigate the dungeon and accrue as many points as possible. To do this, you will be using the *NetHack Learning Environment* [1]. 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 [2])
- A model-based method (e.g. MCTS [3])
- A policy method (e.g. REINFORCE [4])

- A hierarchical method (e.g. Option-Critic [5])

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 then submit your trained agent on Moodle, and it will be evaluated on a number of unseen seeds.

A portion of your mark will consist of a competitive element - your agents' scores will be evaluated and placed on a leaderboard. We will then take the rank of your best agent and use that to determine your mark for this component. The higher your rank, the better your mark will be.

Additionally, you will be submitting a report in which you describe your approach in tackling this problem and give an analysis of your agents' performances. Your report must include the following.

- The names and student numbers of each group member
- A description of each algorithm used.
- 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.

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 *NetHack Learning Environment* can be found on the GitHub repository https://github.com/facebookresearch/nle, along with detailed installation instructions and a brief usage example. The *nle* package comes with a number of environments (described in the arxiv preprint) [1].

The *nle* package comes with several environments. The main environment is the *NetHackScore-v0* environment, which uses the in-game score as the reward signal. Several subtasks are also provided, such as the *NetHackStaircase-v0* environment, which rewards reaching the staircase down to the next level of the

dungeon. You may use these subtasks to train your agents if you wish, but **you will be evaluated using the *NetHackScore-v0* environment**.

By default, the environment sets your character as the **neutral male human monk**. As you will be evaluated using this character, we recommend you do not switch to another character class.

For more information on the environment, including the state and action space, refer to the arxiv preprint [1].

2 Implementation

In order to evaluate your agents, we will be providing a class interface which your agents must use. This interface will consist of two methods: a constructor, in which you can do all of your preprocessing and model loading, and an *act* method, which takes the observed state as input and returns the chosen action. You will need to call any observation wrappers you implement inside this method, as your agents will not have direct access to the environment.

3 Video Submission

To visualise your agents' performance, we will be using the nle dashboard application. Every time you run the *nle* environment, it automatically generates a csv file containing information on each episode, and a ttyrec file for each episode (located inside a zip file). You will be submitting one ttyrec file and one csv file for each agent, corresponding to the episodes you would like to present. The csv files must be renamed to *stats.csv* and the ttyrec files must be renamed to *episode.ttyrec* and zipped into *stats.zip*. Ensure that your csv file only has one row (other than the header) and that the *ttyrec* field contains the correct path to your ttyrec file. See section 5 for an example submission directory tree.

If you wish to view these episodes on your own machine, you can install the dashboard by following the instructions on the dashboard readme. Navigate to the *nle_data* 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.

4 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.

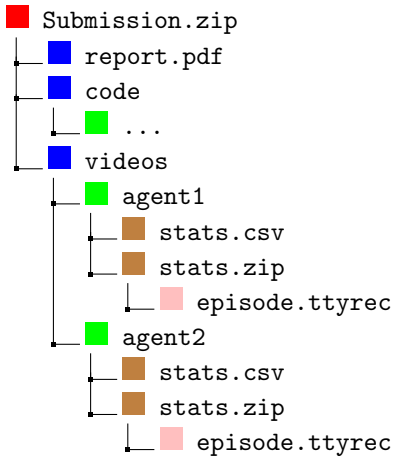
- Your leaderboard ranking

- Your report
  - The descriptions of your chosen methods
  - The descriptions of your design decisions
  - The list of hyperparameters
  - The comparison plot
- Your source code
  - Marks will be awarded for neat, well-commented and reproducible code
- Your videos

5 Submission

For the leaderboard, zip the code which contains your agent classes along with all required files (e.g. your trained models). Ensure the zip file is called *sub0MyAgent.py.zip* and upload it onto Moodle for marking.

For everything else, zip your report, video files and source code and upload it to the relevant Moodle submission page. For each agent, create a separate directory in which *stats.csv* and *stats.zip* (containing *episode.ttyrec*) will be placed. For example, your submission may have the following directory structure.



References

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

[2] 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.

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

[4] 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.

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