Journal of
open research software

SOFTWARE
METAPAPER

A NetHack Learning Environment Language Wrapper for Autonomous Agents

NIKOLAJ GOODGER
PETER VAMPLEW
CAMERON FOALE
RICHARD DAZELEY

*Author affiliations can be found in the back matter of this article

]u[ubiquity press

ABSTRACT

This paper describes a language wrapper for the NetHack Learning Environment (NLE) [1]. The wrapper replaces the non-language observations and actions with comparable language versions. The NLE offers a grand challenge for AI research while MiniHack [2] extends this potential to more specific and configurable tasks. By providing a language interface, we can enable further research on language agents and directly connect language models to a versatile environment.

CORRESPONDING AUTHOR:

Nikolaj Goodger

Federation University Australia
Mt Helen Campus PO Box 663
Ballarat VIC 3353, Australia

ngoodger@protonmail.com

KEYWORDS:

Autonomous Agents; Generalization; Language Models; Language Representations; NetHack; Reinforcement Learning; Transfer Learning

TO CITE THIS ARTICLE:

Goodger N, Vamplew P, Foale C, Dazeley R 2023 A NetHack Learning Environment Language Wrapper for Autonomous Agents. *Journal of Open Research Software,* 11: 8. DOI: https://doi.org/10.5334/jors.444

(1) OVERVIEW

INTRODUCTION

The recent NetHack Learning Environment Challenge [3] established the NetHack Challenge Task within the NetHack Learning Environment (NLE) [1] as a grand challenge for AI. The environment has been extended with MiniHack [2], which features a suite of smaller environments and subtasks for investigating specific behaviors. MiniHack also allows for easily creating new environments using the rich features and dynamics of the NetHack game.

Learning from scratch in NetHack is extremely challenging because of the very large observation space (1000s of different entities in randomized configurations), large action space (greater than 100 distinct compositional actions), and length of the game (>10k steps). During the NetHack Challenge, none of the symbolic, deep-rl, or hybrid agents submitted were able to win the game. An alternative approach is to use the tremendous amount of prior knowledge available in written texts and especially in the NetHackWiki.[1]

Transfer learning has become the standard approach for solving Natural Language Processing (NLP) tasks, and has been used to achieve state-of-the-art results [4]. Sample efficiency can also be improved drastically with Large Language Models (LLM) demonstrating in context few-shot learning [5]. Smaller language models have also been shown to perform few-shot learning with fine-tuning [6]. This makes it attractive to frame a task as a language problem to leverage these models and utilize the knowledge in the pretrained model. Pretrained models have been successfully applied to Reinforcement Learning (RL) [7], Interactive Decision-Making [8], and Instruction Following [9]. However, these examples still utilize non-language elements with additional parameters that require training from scratch, precluding few-shot levels of sample efficiency.

For common sense reasoning, language models trained on natural language do not perform as well as humans [10]. One approach to improving the common sense reasoning capability of language models is by interacting with an environment to gather knowledge not normally written down. For example, if I put an item into a bag I can take it out again later.

The use of language representations in the observation and action space can improve generalization when compared to vector observations [11]. This is likely because of the compositional structure of language. Language representations can also be used to describe all observation modalities, allowing a proven language model architecture, like the transformer [12] to be used without the need for bespoke multi-modal models.

In this paper, we present a wrapper for the NLE that uses language to encode the non-textual observations and similarly decode language actions to the supported discrete actions. To summarize the main reasons for choosing to create a language interface are as follows:

1. A pretrained language model can be directly connected with the environment and so knowledge learned during language model pretraining can transfer to the agents task, improving sample efficiency.
2. Language modeling alone does not always work for learning common sense. By embodying the agent the language model would need to learn common sense to achieve its goals.
3. By using compositional language observations and actions we enable agents capable of compositional generalization.
4. Exclusively using the language modality for observations and actions allows for adding additional features, new actions, new goals, and outputting explanations without changing the model architecture.
5. To facilitate further research on language agents.

RELATED WORK

Some of the earliest work on applying deep reinforcement learning to text-based games [13, 14] use the Evennia Game Engine[2] and test on existing games or developed new ones to facilitate their research. The TextWorld environment [15] is designed to test transfer learning and generalization on different language-based games. However, TextWorld lacks the complexity of real games, because of its limited quest generation capability. TextWorld is also slow to run, but this particular limitation was entirely mitigated in TextWorldExpress [16], however the environments are still limited in complexity. Another text environment is WordCraft [17], combining constituent entities in the presence of distractors to produce a goal entity which tests common-sense knowledge but also intentionally limits the scope of the environment.

IMPLEMENTATION AND ARCHITECTURE

To translate the NLE non-language observations to language representations and language actions to the keyboard actions of the environment we wrap the base NLE. This process is visualized in the block diagram in Figure 1. The space of language interpretations of NLE observations is large, so we define some design goals to target the key functionality required to solve the problem:

1. Include key entities, e.g. the agent must know about monsters so that it can attack or escape.
2. Allow navigation. To be able to navigate the agent should be able to see obstacles and find a path to its goal.
3. Allow effective usage of ranged tools. Ranged weapons are obstructed by obstacles and ray

weapons can be reflected. So the agent must know the position of nearby obstacles.

4. Performant. The NLE environment is extremely fast making it useful for RL experiments, and we want to minimize the overhead when adding the wrapper

LANGUAGE ENCODING FOR VISUAL ELEMENTS

In the NLE the screen observations include glyphs, which are integers that uniquely represent all the objects in the game. The screen is 79 × 21 for a total of 1659 glyphs. To convert this into a serialized language representation we use the *subject complement* grammar. We maintain an array of glyph strings, so for each integer glyph, we can look up the corresponding string entity with O(1) efficiency. The entity is used as the *subject*, we then add the distance and direction relative to the agent as its *subject complement* forming a triple of *entity*, *distance*, and *direction*, e.g. {"giant ant", "far", "northeast"} (see Figure 2a for the flow). Performing this operation for each glyph yields a collection of triples. These triples go through a number of post-processing steps shown and described in Figure 2b to produce a language observation.

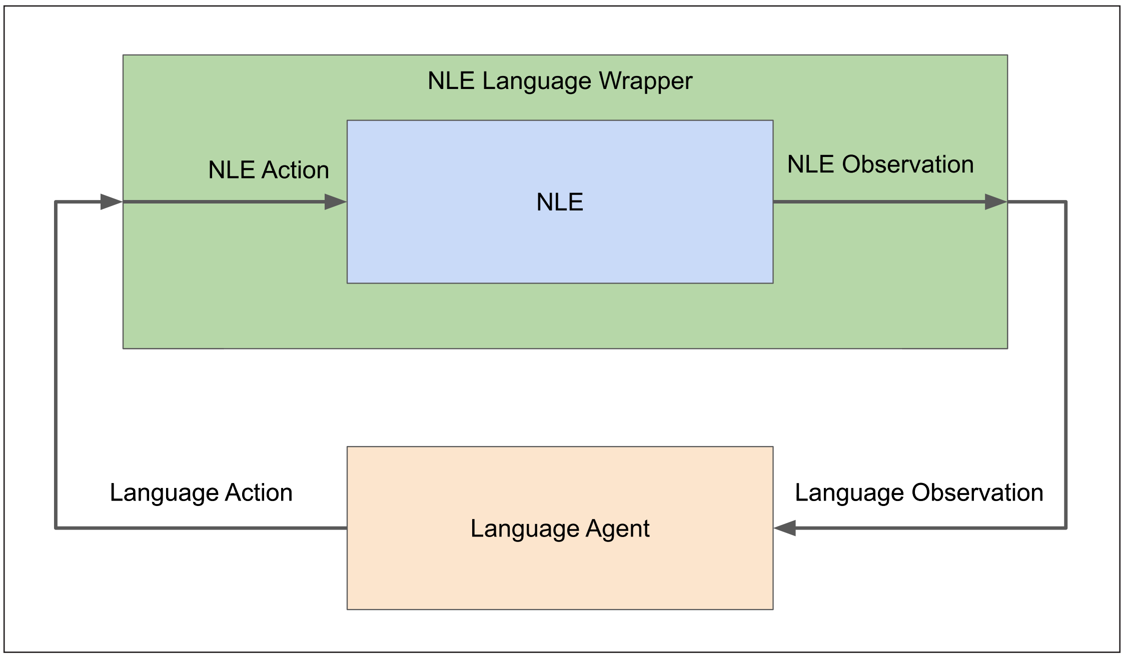


Figure 1 NLE Language Wrapper Block Diagram.

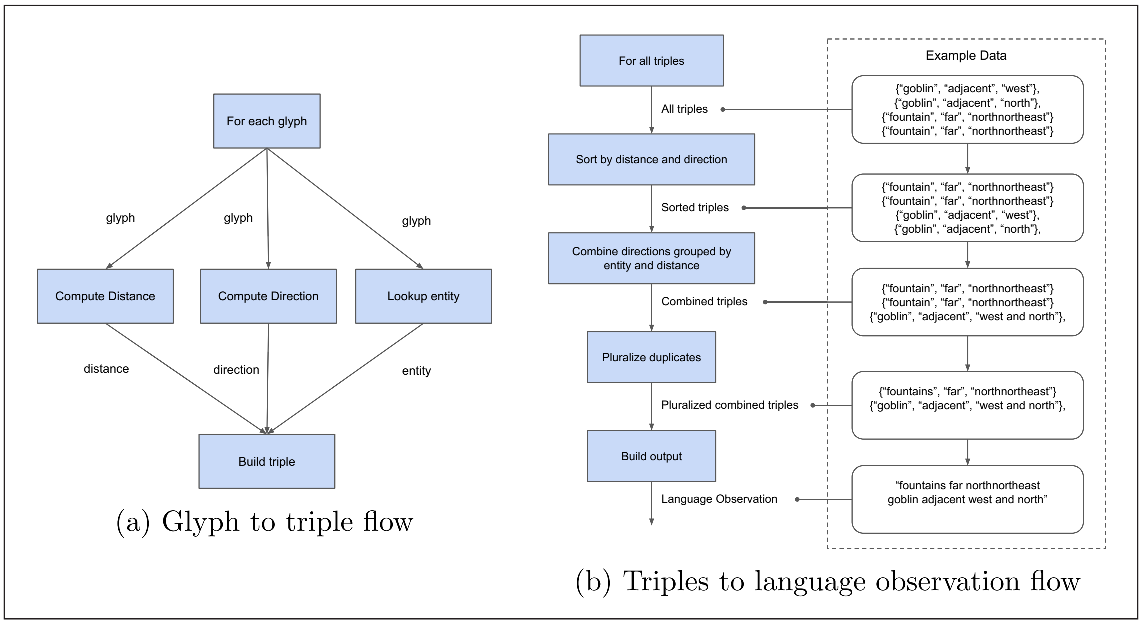

For each glyph
glyph
glyph
glyph
Compute Distance
Compute Direction
Lookup entity
distance
direction
entity
Build triple
(a) Glyph to triple flow
For all triples
All triples
Sort by distance and direction
Sorted triples
Combine directions grouped by entity and distance
Combined triples
Pluralize duplicates
Pluralized combined triples
Build output
Language Observation
Example Data
{"goblin", "adjacent", "west"}, {"goblin", "adjacent", "north"}, {"fountain", "far", "northnortheast"} {"fountain", "far", "northnortheast"}
{"fountain", "far", "northnortheast"} {"fountain", "far", "northnortheast"} {"goblin", "adjacent", "west"}, {"goblin", "adjacent", "north"},
{"fountain", "far", "northnortheast"} {"fountain", "far", "northnortheast"} {"goblin", "adjacent", "west and north"},
{"fountains", "far", "northnortheast"} {"goblin", "adjacent", "west and north"},
"fountains far northnortheast goblin adjacent west and north"
(b) Triples to language observation flow

Figure 2 These figures show the computation flow for converting glyphs to the language observation. Figure 2a shows how for each glyph we compute the distance, direction and entity strings to produce a triple. Figure 2b shows the mapping for the collection of triples to the language observation, with example data on the right-hand side.

The distances are quantized into buckets of *adjacent*, *very near*, *near*, *far*, and *very far*. The directions are also quantized into cardinal, inter-cardinal, and intermediate directions, e.g. *north*, *northeast*, and *northnortheast*. See Figure 3 for a visualization of how the distances and directions are calculated. A complete example observation is shown in Figure 4.

VISUAL VIEW

Despite utilizing a compact language representation the screen size of 1659 means that an exhaustive description of every glyph would be enormous. To address this we draw inspiration from the unconscious and conscious bandwidth of the brain. The eyes process orders of magnitude more information unconsciously compared to what can be read consciously [18]. The limited information throughput when reading compared to a visual input implies that for the representations to be interpreted in a similar time frame (at least for a human) we should keep only the salient information and discard the rest. Discarding information in the input and reducing the environment's observability will negatively impact the performance of an optimal agent. However, the objective of this work is to build an environment that enables the agent to find a solution, not necessarily the optimal one.

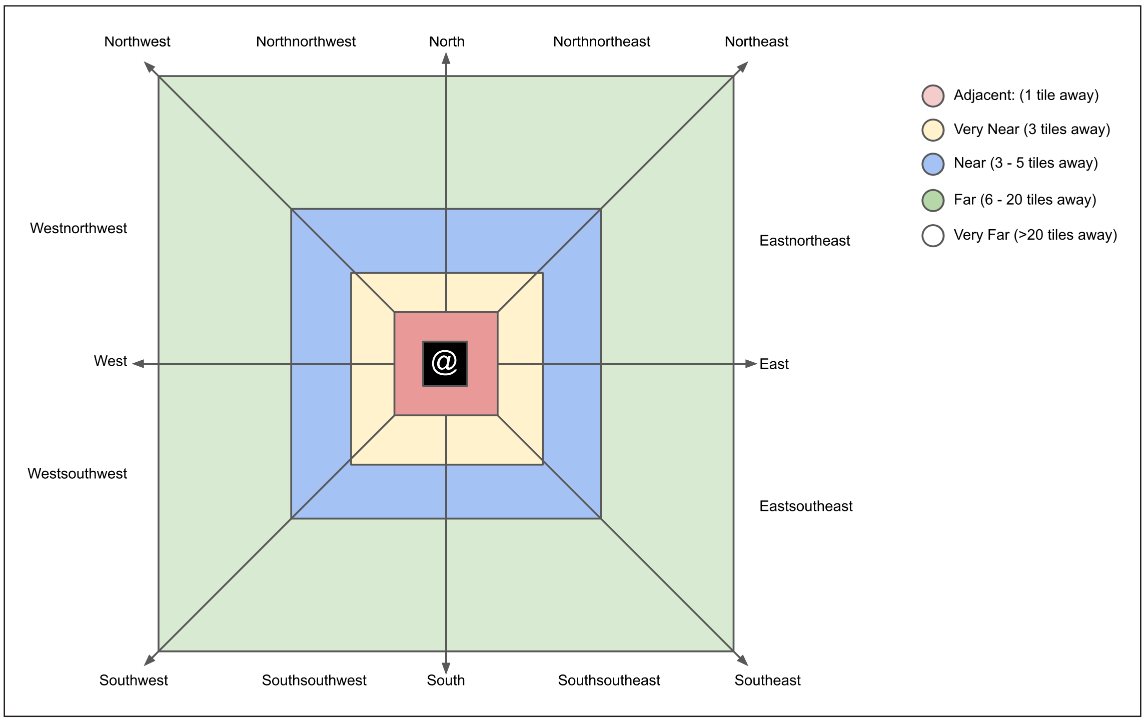
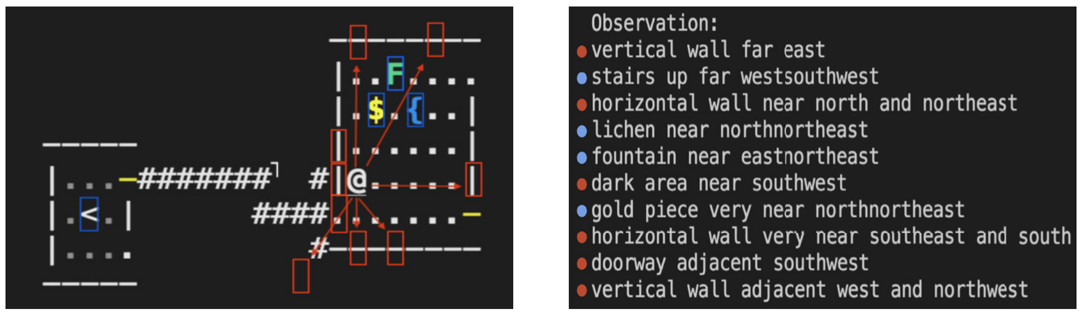


Figure 3 In this figure, the player is represented by the @ symbol in the center. The color bands around the player represent distances, that are defined in the legend on the right. Glyphs that lie on cardinal or inter-cardinal directions (e.g. north, northeast, east) are defined as being in the direction they lie on. Glyphs that are located between the cardinal and inter-cardinal directions are assigned the direction of the direction band they fall within (e.g. northnortheast), regardless of their exact position within the band. Using this chart we can see how for any glyph position, the distances and directions can be quantized to text relative to the player.



```
Observation:
●vertical wall far east
●stairs up far westsouthwest
●horizontal wall near north and northeast
●lichen near northnortheast
●fountain near eastnortheast
●dark area near southwest
●gold piece very near northnortheast
●horizontal wall very near southeast and south
●doorway adjacent southwest
●vertical wall adjacent west and northwest
```

(a) NetHack Screen (b) Language Observation

Figure 4 The screen in Figure 4a shows the glyphs included in the language observation listed in Figure 4b. The elements and rays from the Visual View are indicated with red, and the elements from the Fullscreen View are indicated with blue.

Discarding unnecessary information is achieved using Views, which capture specific glyphs based on rules. Currently, the wrapper has two views defined, the Fullscreen View and the Visual View. It is possible to add more Views by following the same design pattern.

FULLSCREEN VIEW

This View captures all glyphs that are not floors, walls, or unexplored tiles. This process discards the majority of the information on the screen but retains the important items of interest, such as monsters, items, waypoints, etc. Using the Fullscreen View alone may potentially be enough to make progress on the game, but the exclusion of obstacles has disadvantages the Visual View tries to address.

One problem with the Fullscreen View is the lack of information on obstacles. This makes navigation more difficult. Another challenge is some actions can be impacted by obstacles, e.g. ray spells can be reflected by obstacles and hit the agent. To address this problem we try to include the key obstacles in the language observation using the Visual View. Here we simulate vision for the agent in the cardinal and inter-cardinal directions (The only directions the agent can directly interact in). By implementing a simple ray marching system we cast a ray that stops when it encounters a glyph that is either blocking vision or cannot be perceived, which can occur as NetHack also simulates a field of view. This ray reports any glyph that is not a floor and is mutually exclusive to the Fullscreen View (to avoid duplicates). This approach offers the agent some basic navigational queues and allows for safe usage of ray spells. To solve navigational tasks the agent will need to be more reliant on memory or apply a simpler strategy like wall following compared to a multi-modal agent using the raw NLE input.

VECTOR OBSERVATIONS

The vector observation or bottom-line stats (blstats) includes useful features like hitpoints, gold, hunger, etc. To textualize this feature we create pairs of one or more vector features using the template [*label*]:[*value*], e.g. the vector values for current hitpoints and maximum hitpoints becomes "HP: 12/24". When possible we encode the vector value using text, e.g. hunger values 0, 1, 2 become "Satiated", "Not Hungry" and "Hungry" which are the same as the in-game representations of these states. An example observation for hunger is "Hunger: Satiated".

LANGUAGE ENCODING ACTIONS

There is a unique action in NetHack assigned to each keyboard button and additional actions are available by pressing modifier keys Ctrl and Shift. These actions also have names consisting of one or more words like apply, north, dip, etc. So using a language action space it is natural to assign these words to the actions. However, because many of the actions can be composed, the action definition can change, e.g. the *a* key refers to the "apply" action, so we map the action string "apply" to that key, but it can also refer to the first item in the agent's inventory, so we might perform a composed action like "eat" "a". Therefore, we also include a mapping of the action string "a" to the keyboard button *a* so both semantic language actions are available to the agent. Invalid actions raise a *ValueError* Exception which must be handled by the agent.

SCALABILITY

The NLE is comparatively fast compared to other environments, running at 14.4k steps per second on an Intel Core i7 2.9 GHz CPU [1]. To implement this wrapper we require extensive string manipulation logic. This is compute intensive, so we implement the transformations in C++ using PyBind.[3] We compare the environment Steps Per Second (SPS) running on a Ryzen 1700 CPU in Table 1 by taking random actions for 10k steps and taking the average of 3 runs. The results show that despite the complexity of the string manipulation, the wrapper retains 40% of the performance of NLE which is enough to run RL experiments. We have also implemented equivalent performance integration tests in the test suite to avoid regression when refactoring or adding new features.

EXPERIMENTS

To validate the wrapper a Sample Factory [19] implementation is included. This uses Asynchronous Proximal Policy Optimization (APPO) to optimize an agent online. The Agent uses the huggingface transformer Library [20] for a policy and value function model. As a baseline, for this implementation, we used the nle-sample-factory-baseline.[4] The results are shown in Table 2.

QUALITY CONTROL

The wrapper includes integration tests to validate key functionality. It also includes performance tests to prevent regression during further development. All the tests are listed in Table 3. The environment has been

ENVIRONMENT	SPS
NLE	15k
NLE with Language Wrapper	6k

Table 1 NLE Language Wrapper Performance.

EXPERIMENT	AVG REWARD
sample factory baseline	566 (16)
language wrapper	695 (22)

Table 2 Average reward Mean (Standard Deviation) of 3 runs for 1B steps.

TEST NAME	RESULT
test_message_spell_menu	PASSED
test_message_more_end	PASSED
test_message_full_stop_end	PASSED
test_message_bracket_end	PASSED
test_message_parenthesis_end	PASSED
test_message_multipage	PASSED
test_message_takeoffall	PASSED
test_filter_map_from_conduct	PASSED
test_empty_tty_chars_returns_empty_message	PASSED
test_filter_map_from_name	PASSED
test_filter_map_travel	PASSED
test_create_env_real	PASSED
test_env_language_action_space	PASSED
test_env_discrete_action_space	PASSED
test_env_obsv_space	PASSED
test_step_real	PASSED
test_step_invalid_action	PASSED
test_action_actions_maps_reflect_valid_actions	PASSED
test_step_valid_action_not_supported	PASSED
test_obsv_fake	PASSED
test_blstats_condition_none	PASSED
test_blstats_condition_flying	PASSED
test_multiple_obsv_fake	PASSED
test_step_fake	PASSED
test_statue	PASSED
test_warning	PASSED
test_swallow	PASSED
test_zap_beam	PASSED
test_explosion	PASSED
test_illegal_object	PASSED
test_weapon	PASSED
test_armour	PASSED
test_ring	PASSED
test_amulet	PASSED
test_tool	PASSED
test_food	PASSED
test_potion	PASSED
test_scroll	PASSED
test_spellbook	PASSED
test_wand	PASSED
test_coin	PASSED

(Contd.)

TEST NAME	RESULT
test_gem	PASSED
test_rock	PASSED
test_ball	PASSED
test_chain	PASSED
test_venom	PASSED
test_ridden	PASSED
test_corpse	PASSED
test_invisible	PASSED
test_detected	PASSED
test_tame	PASSED
test_monster	PASSED
test_plural_end_ey	PASSED
test_plural_end_y	PASSED
test_plural_default	PASSED
test_plural_end_s	PASSED
test_plural_end_f	PASSED
test_plural_end_ff	PASSED
test_plural_lava	PASSED
test_wrapper_only_works_with_nle_envs	PASSED
test_wrapper_requires_all_keys	PASSED
test_play	PASSED
test_time_reset	PASSED
test_time_step	PASSED

Table 3 NLE Language Wrapper integration tests.

validated at scale while training the included agent for 1 billion steps. Detailed explanations and examples of how to run and test the environment are documented in the README.

(2) AVAILABILITY

OPERATING SYSTEM

MacOS, Linux and Windows using WSL.

PROGRAMMING LANGUAGE

Python 3.7 or higher.

ADDITIONAL SYSTEM REQUIREMENTS

None

DEPENDENCIES

See Table 4 for a list of the project dependencies.

SOFTWARE LOCATION

Code repository GitHub

Name: https://github.com/ngoodger/nle-language-wrapper

DOI: https://www.doi.org/10.5281/zenodo.7456086

Licence: MIT License

Date published: 04/07/22

LANGUAGE

English

(3) REUSE POTENTIAL

As an interactive environment using the de facto standard OpenAI gym interface, this wrapper is specifically directly suited for developing online RL algorithms using language models. It could also be used for Offline RL, Imitation learning, and Decision-Making research, if additional work is done to record and save trajectories from a policy. Other possibilities for re-use include extensions or forking of the wrapper, which is fully permitted under the MIT License, and may be useful

COMPONENT	DEPENDENCY	FUNCTION
base	gym>=0.15, gym<=0.23	Wrapper base class
base	minihack>=0.1.3	Enable wrapper for MiniHack
base	nle==0.8.1	Base environment
base	pybind11>=2.9	Implement high performance functions
dev	black>=22.6.0	Formatting Python
dev	flake8>=4.0.1	Linting Python
dev	pytest>=7.1.2	Test framework
dev	pytest-cov>=3.0.0	Test coverage
dev	pytest-mock>=3.7.0	Test mocks
dev	pygame>=2.1.2	Used for specific test
dev	isort>=5.10.1	Sort dependencies
dev	numpy>=1.21.0	Used for test framework
agent	sample_factory>=1.121.4	RL framework
agent	transformers>=4.17.0	Language model for agent

Table 4 List of core dependencies for the wrapper. The **Component** column identifies what the dependency is required for, where those marked *base* are required to use the wrapper, *dev* are required for development, and *agent* are required to train or run the included sample factory agent. The **Dependency** column specifies the library name and the version. Finally, the **Function** column specifies the role of the library in the project.

if users wish to modify some or all of the functionality. Feedback or contributions are welcome and can be made by raising GitHub Issues or Pull-Requests against the repository. Support can also be obtained by raising a GitHub Issue.

NOTES

1 NetHackWiki https://nethackwiki.com/.
2 Evennia Game Engine https://evennia.com.
3 https://github.com/pybind/pybind11.
4 https://github.com/Miffyli/nle-sample-factory-baseline.

ACKNOWLEDGEMENTS

We would like to acknowledge Anssi Kanervisto and Karolis Ramanauskas for sharing their sample-factory implementation.

COMPETING INTERESTS

The authors have no competing interests to declare.

AUTHOR AFFILIATIONS

Nikolaj Goodger orcid.org/0009-0006-7214-0979
Federation University Australia Mt Helen Campus PO Box 663 Ballarat VIC 3353, Australia

Peter Vamplew orcid.org/0000-0002-8687-4424
Federation University Australia Mt Helen Campus PO Box 663 Ballarat VIC 3353, Australia

Cameron Foale orcid.org/0000-0003-2537-0326
Federation University Australia Mt Helen Campus PO Box 663 Ballarat VIC 3353, Australia

Richard Dazeley orcid.org/0000-0002-6199-9685
Deakin University - Locked Bag 20000, Geelong, VIC 3220, AU

REFERENCES

1. **Küttler H, Nardelli N, Miller A, Raileanu R, Selvatici M, Grefenstette E, Rocktäschel T.** The nethack learning environment. In Larochelle H, Ranzato M, Hadsell R, Balcan MF, Lin H (eds.), *Advances in Neural Information Processing Systems.* 2020; 33: 7671–7684. Curran Associates, Inc. URL https://proceedings.neurips.cc/paper/2020/file/569ff987c643b4bedf504efda8f786c2-Paper.pdf.
2. **Samvelyan M, Kirk R, Kurin V, Parker-Holder J, Jiang M, Hambro E, Petroni F, Kuttler H, Grefenstette E, Rocktäschel T.** Minihack the planet: A sandbox for open-ended reinforcement learning research. In Vanschoren J, Yeung S (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks.* 2021; 1. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/file/fa7cdfad1a5aaf8370ebeda47a1ff1c3-Paper-round1.pdf.
3. **Hambro E, Mohanty S, Babaev D, Byeon M, Chakraborty D, Grefenstette E, Jiang M, Daejin J, Kanervisto A, Kim**

J, Kim S, Kirk R, Kurin V, Küttler H, Kwon T, Lee D, Mella V, Nardelli N, Nazarov I, Ovsov N, Holder J, Raileanu R, Ramanauskas K, Rocktäschel T, Rothermel D, Samvelyan M, Sorokin D, Sypetkowski M, Sypetkowski M. Insights from the neurips 2021 nethack challenge. In Kiela D, Ciccone M, Caputo B (eds.), *Proceedings of the NeurIPS 2021 Competitions and Demonstrations Track,* volume 176 of *Proceedings of Machine Learning Research.* 06–14 Dec 2022; 41–52. PMLR. URL https://proceedings.mlr.press/v176/hambro22a.html.

4. **Ruder S, Peters ME, Swayamdipta S, Wolf T.** Transfer learning in natural language processing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials.* June 2019; 15–18. Minneapolis, Minnesota: Association for Computational Linguistics. URL https://aclanthology.org/N19-5004. DOI: https://doi.org/10.18653/v1/N19-5004
5. **Brown T, Mann B, Ryder N, Subbiah M, Kaplan JD, Dhariwal P, Neelakantan A, Shyam P, Sastry G, Askell A, Agarwal S, Herbert-Voss A, Krueger G, Henighan T, Child R, Ramesh A, Ziegler D, Wu J, Winter C, Hesse C, Chen M, Sigler E, Litwin M, Gray S, Chess B, Clark J, Berner C, McCandlish S, Radford A, Sutskever I, Amodei D.** Language models are few-shot learners. In Larochelle H, Ranzato M, Hadsell R, Balcan MF, Lin H (eds.), *Advances in Neural Information Processing Systems.* 2020; 33: 1877–1901. Curran Associates, Inc. URL https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.
6. **Schick, T, Schütze H.** It's not just size that matters: Small language models are also few-shot learners. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.* June 2021; 2339–2352. Online. Association for Computational Linguistics. URL https://aclanthology.org/2021.naacl-main.185. DOI: https://doi.org/10.18653/v1/2021.naacl-main.185
7. **Reid M, Yamada Y, Gu SS.** Can wikipedia help offline reinforcement learning? 2022. URL https://arxiv.org/abs/2201.12122.
8. **Li S, Puig X, Paxton C, Du Y, Wang C, Fan L, Chen T, Huang D-A, Akyürek E, Anandkumar A, Andreas J, Mordatch I, Torralba A, Zhu Y.** Pre-Trained Language Models for Interactive Decision-Making. *arXiv e-prints*, art. arXiv:2202.01771; February 2022.
9. **Hill F, Mokra S, Wong N, Harley T.** Human instruction-following with deep reinforcement learning via transfer-learning from text; 2020. URL https://arxiv.org/abs/2005.09382.
10. **Zhou X, Zhang Y, Cui L, Huang D.** Evaluating commonsense in pre-trained language models; 2019. URL https://arxiv.org/abs/1911.11931.
11. **Goodger N, Vamplew P, Foale C, Dazeley R.** Language representations for generalization in reinforcement learning. In Vineeth NB, Ivor T, (eds.), *Proceedings of The 13th Asian Conference on Machine Learning.* 2021; 157: 390–405. Virtual.
12. **Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Ukasz Kaiser Ł, Polosukhin I.** Attention is all you need. In Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S, Garnett R (eds.), *Advances in Neural Information Processing Systems.* 2017; 30. Curran Associates, Inc. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.
13. **Narasimhan K, Kulkarni T, Barzilay R.** Language understanding for text-based games using deep reinforcement learning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing.* September 2015; 1–11. Lisbon, Portugal: Association for Computational Linguistics. DOI: https://doi.org/10.18653/v1/D15-1001
14. **He J, Chen J, He X, Gao J, Li L, Deng L, Ostendorf M.** Deep reinforcement learning with a natural language action space. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers).* August 2016; 1621–1630. Berlin, Germany: Association for Computational Linguistics. URL https://aclanthology.org/P16-1153. DOI: https://doi.org/10.18653/v1/P16-1153
15. **Côté M-A, Kádár Á, Yuan X, Kybartas B, Barnes T, Fine E, Moore J, Tao RY, Hausknecht M, El Asri L, Adada M, Tay W, Trischler A.** Textworld: A learning environment for text-based games. *CoRR,* abs/1806.11532; 2018. DOI: https://doi.org/10.1007/978-3-030-24337-1_3
16. **Jansen PA, Côté M.** Textworldexpress: Simulating text games at one million steps per second. *CoRR,* abs/2208.01174; 2022. DOI: https://doi.org/10.48550/arXiv.2208.01174
17. **Jiang M, Luketina J, Nardelli N, Minervini P, Torr PHS, Whiteson S, Rocktäschel T.** Wordcraft: An environment for benchmarking commonsense agents. In *Workshop on Language in Reinforcement Learning (LaRel)*; 2020. URL https://github.com/minqi/wordcraft.
18. **Koch K, McLean J, Segev R, Freed M, Berry II M, Balasubramanian V, Sterling P.** How much the eye tells the brain. *Current biology: CB.* 08 2006; 16: 1428–34. DOI: https://doi.org/10.1016/j.cub.2006.05.056
19. **Petrenko A, Huang Z, Kumar T, Sukhatme G, Koltun V.** Sample factory: Egocentric 3d control from pixels at 100000 fps with asynchronous reinforcement learning. In *ICML*; 2020. DOI: https://doi.org/10.1016/j.cub.2006.05.056
20. **Wolf T, Debut L, Sanh V, Chaumond J, Delangue C, Moi A, Cistac P, Rault T, Louf R, Funtowicz M, Davison J, Shleifer S, von Platen P, Ma C, Jernite Y, Plu J, Xu C, Le Scao T, Gugger S, Drame M, Lhoest Q, Alexander M.** Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations.* October 2020; 38–45. Online. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6. DOI: https://doi.org/10.18653/v1/2020.emnlp-demos.6

TO CITE THIS ARTICLE:
Goodger N, Vamplew P, Foale C, Dazeley R 2023 A NetHack Learning Environment Language Wrapper for Autonomous Agents. *Journal of Open Research Software,* 11: 8. DOI: https://doi.org/10.5334/jors.444

Submitted: 25 October 2022 **Accepted:** 16 March 2023 **Published:** 13 June 2023