Insights from the NeurIPS 2021 NetHack Challenge

Eric Hambro[*+] EHAMBRO@FB.COM
Sharada Mohanty[*=] MOHANTY@AICROWD.COM
Dmitrii Babaev[#?] BABAEV@AIRI.NET
Minwoo Byeon[$] MINU@KAKAOBRAIN.COM
Dipam Chakraborty[=] DIPAM@AICROWD.COM
Edward Grefenstette[+!] EGREFEN@FB.COM
Minqi Jiang[+!] MSJ@FB.COM
Daejin Jo[$] DAEJIN.JO@KAKAOBRAIN.COM
Anssi Kanervisto[<] ANSSI.KANERVISTO@UEF.FI
Jongmin Kim[$] JMKIM@KAKAOBRAIN.COM
Sungwoong Kim[$] SWKIM@KAKAOBRAIN.COM
Robert Kirk[!] ROBERT.KIRK.20@UCL.AC.UK
Vitaly Kurin["] VITALY.KURIN@CS.OX.AC.UK
Heinrich Küttler[+] HNR@FB.COM
Taehwon Kwon[$] TAEHWAN.KWON@KAKAOBRAIN.COM
Donghoon Lee[$] DHLEE@KAKAOBRAIN.COM
Vegard Mella[+] VEGARDMELLA@FB.COM
Nantas Nardelli["] NANTAS.NARDELLI@GMAIL.COM
Ivan Nazarov[?] NAZAROV@AIRI.NET
Nikita Ovsov[#] OVSOV.N.P@SBERBANK.RU
Jack Parker-Holder["] JACKPH@ROBOTS.OX.AC.UK
Roberta Raileanu[+] RAILEANU@FB.COM
Karolis Ramanauskas[−] KAROLIS.RAM@GMAIL.COM
Tim Rocktäschel[+!] ROCKT@FB.COM
Danielle Rothermel[+] DROTHERM@FB.COM
Mikayel Samvelyan[+!] SAMVELYAN@FB.COM
Dmitry Sorokin[?] SOROKIN@AIRI.NET
Maciej Sypetkowski[−] MACIEJ.SYPETKOWSKI@GMAIL.COM
Michał Sypetkowski[−] M.SYPETKOWSKI@GMAIL.COM

Editor: Douwe Kiela, Barbara Caputo, and Marco Ciccone

Abstract

In this report, we summarize the takeaways from the first NeurIPS 2021 NetHack Challenge. Participants were tasked with developing a program or agent that can win (i.e., 'ascend' in) the popular dungeon-crawler game of NetHack by interacting with the NetHack Learning Environment (`NLE`), a scalable, procedurally generated, and challenging *Gym* environment for reinforcement learning (RL). The challenge showcased community-driven progress in AI with many diverse approaches significantly beating the previously best results on NetHack. Furthermore, it served as a direct comparison between neural (e.g., deep RL) and symbolic AI, as well as hybrid systems, demonstrating that on NetHack symbolic bots currently outperform deep RL by a large margin. Lastly, no agent got close to winning the game, illustrating NetHack's suitability as a long-term benchmark for AI research.

Keywords: Reinforcement Learning, Open-ended Learning, Generalization, Procgen, Game AI, NetHack

1. Introduction

Progress in artificial intelligence research requires benchmarks that test the limits of current methods. To this end, the NeurIPS 2021 NetHack Challenge was a competition to drive the open benchmarking of current sequential decision-making methods on the NetHack Learning Environment (`NLE`, Küttler et al., 2020). `NLE` is a fully-featured *Gym* environment (Brockman et al. (2016)) based on the popular open-source terminal-based single-player procedurally-generated "dungeon-crawler" game, NetHack (Raymond (1987); Kenneth Lorber (2020)).

Aside from procedurally generated content, NetHack is an attractive research platform as it contains hundreds of enemy and object types, has complex and stochastic environment dynamics, and has a clearly defined goal (descend the dungeon, retrieve an amulet, and ascend) which can be achieved in a diverse set of ways. The game is considered one of the hardest in the world[1], with winning episodes lasting 100,000s of steps, and a permadeath setting that starts agents at the beginning in a whole new world if they die in the dungeon. NetHack is even difficult to master for human players who often rely on external knowledge, such as the many extensive community-created documents outlining various strategies for the game (NetHack Wiki (2020); Eva Myers (2020)), to learn about strategies and NetHack's complex dynamics and secrets.

NetHack has a long history of online tournaments, often played on various competition servers. The longest running of these is the `/dev/null/nethack` tournament[2] which ran every year from 1999 to 2016 and has now been superseded by the November NetHack Tournament.[3] While bots are forbidden from entering this tournament, the community still has a large history of creating symbolic bots, such as SWAGGINZZZ (2019), TAEB (2015), BotHack (2014). In the past, these bots have often made progress by exploiting bugs or weaknesses in the earlier versions of the game that have since been removed by the game developers. To the best of our knowledge, no bot has ever ascended in the most recent version of the game, NetHack 3.6.6.

At NeurIPS 2021, we challenged participants to either train a machine learning agent or hard-code a symbolic bot to win the game of NetHack. In this report, we present in-depth analyses of the main results of this challenge, in particular i) the community-driven progress in AI for NetHack compared to the previous state-of-the-art, ii) the current dominance of symbolic bots over deep RL approaches, and iii) the difficulty and complexity of NetHack leading to none of the participating agents getting close to winning the game.

2. Competition

In this section, we describe the `NLE` environment to which participants had access to, as well as the evaluation metrics and competition structure.

*Lead organizers. Author ordering for the other authors is alphabetical. +Facebook AI Research =AIcrowd !University College London "University of Oxford #Sberbank AI Lab ?AIRI $KakaoBrain <University of Eastern Finland ¯Independent

1. https://www.telegraph.co.uk/gaming/what-to-play/the-15-hardest-video-games-ever/nethack/
2. https://nethackwiki.com/wiki//dev/null/nethack_tournament
3. https://www.hardfought.org/tnnt/

2.1. Environment

We created the new `NLE` task, "`challenge`", for the competition, in order to expose the full game of NetHack 3.6.6 in all its complexity. This environment expanded the action space from 23 to 113 actions (the full keyboard), removed the autoclosing of popups, and allowed all forms of input modality. It expanded the observation space to be as broad as possible, including structured observations like `glyphs`, `message` and `blstats`, inventory observations like `inv_strs` and `inv_glyphs`, as well as the raw terminal outputs in `ttychars`, `ttycolors`, `ttycursor` which is what human players have access to when playing NetHack. Finally, this environment enforced rotation of the starting character's race, role, alignment and gender. This in particular presented a much harder challenge to competitors than the previous character-specific NetHack score task in (Küttler et al., 2020), thus incentivising the development of general agents instead of character-specific approaches.

We released `challenge` in `NLE` v0.7.0, and improved it with minor bug fixes over the first month of the challenge, before freezing at v0.7.3. All evaluations were run on v0.7.3.

2.2. Metrics

Submission performance was measured with a tuple of three statistics calculated over the test episodes. These statistics were, in order of tie-breaking: number of ascensions, median in-game score, and mean in-game score. While the in-game score does not directly correspond to making progress towards ascension, it does generally correspond to good play. As evidence, the minimum score needed for ascension is 12,200, but the average ascending run score is 6.98 million.[4] However, this correlation does not always hold—some expert players attempt to ascend while, in fact, minimizing the score. Full details of the score methodology can be found on the NetHack Wiki.[5]

2.3. Competition Structure

Thanks to our sponsors, Meta AI and DeepMind, the competition was able to award $20,000 in prizes to competitors, across 4 different tracks. These tracks were: Best Overall Agent, Best Neural Agent,[6] Best Symbolic Agent,[7] and Best Agent from an Academic Lab/Independent Researcher(s). This was designed to incentivise a showdown between symbolic and deep RL methods since Tracks 2 and 3 were mutually exclusive, but otherwise, single teams could win multiple tracks. Given the suitability of `NLE` as a "computationally cheap" Grand Challenge, we also sought to incentivise submissions from independent researchers or academic labs.

The competition was divided into two phases: a development phase, running June to October, and a test phase, for the last two weeks of October. Only the top 15 entries for each track qualified for the test phase. In the development phase, we evaluated on 512 episode rollouts, within a 2-hour window, and in the test phase, three submissions were evaluated on 4,096 episode rollouts, over a day. This allowed competitors to evaluate different approaches

4. Calculated from a sample of 20,000 ascending human runs from alt.org
5. https://nethackwiki.com/wiki/Score
6. Referred to as "Best Agent Making Substantial Use of a Neural Network" on aicrowd.com
7. Referred to as "Best Agent **Not** Making Use of a Neural Network" on aicrowd.com

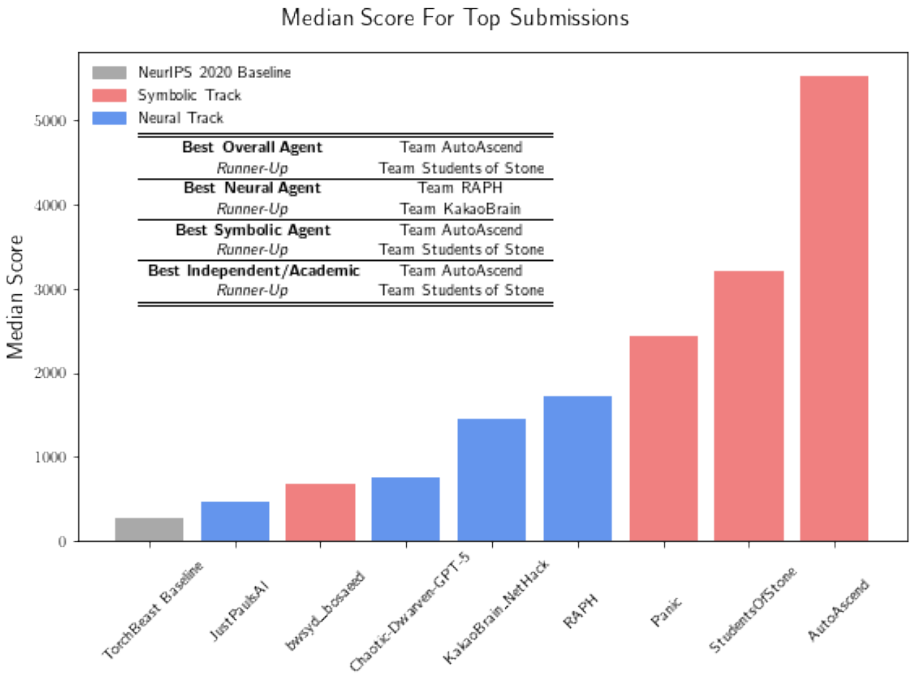

Figure 1: Final Rankings of Top 8 Teams

over the bulk of the competition but allowed us to whittle down competitors to a manageable number for a thorough final evaluation.

2.4. Evaluation Setup

The AIcrowd platform was the technological centre of the competition. AIcrowd hosted a starter kit on their GitLab instance, with a TorchBeast (2019) training pipeline, two pre-trained models, and a submission pipeline. They also hosted the leaderboard, a discussion forum and demo Colab notebooks. Most crucially they hosted an automated evaluation service and provided team members with feedback on their performance. More details on the evaluation setup can be found in Appendix A.

3. Results & Discussion

In this section, we summarize and discuss the results of the NeurIPS 2021 NetHack Challenge. Overall, the NetHack Challenge ran for 144 days, with 483 entrants registering for the competition, and 631 submissions received over that time.

3.1. Competition Results

As mentioned in Section 2, the competition was divided into two phases. In Phase 1, 28 entrants beat the official IMPALA baseline, and 17 passed through to Phase 2. Of these Phase 2 entrants, 8 entrants beat the IMPALA baseline by a significant margin (33%+), and the rankings and prizewinners are shown in Figure 1.

The results demonstrate that symbolic approaches currently significantly outperform machine learning, and in particular deep RL-based agents. The top three places in the Overall Best Agent all went to agents from the Symbolic Agent Track, and the winning Neural Agent (RAPH) was a hybrid model, making heavy use of symbolic methods and a neural network. Symbolic bots outperformed other approaches by a significant margin: the best symbolic bot's median score was nearly threefold that of the neural bot's. This lead extends fivefold when looking at the top-scoring episode from each track. Nonetheless, deep reinforcement learning bots made significant progress, offering a nearly fivefold improvement in median score over our NeurIPS 2020 baseline (Küttler et al., 2020). Overall, these results demonstrate significant community-driven progress on AI for NetHack.

3.2. Analyzing Agent Behavior

To get a better understanding of the results, we analysed the trajectories of all three "final round" submissions from the Phase 2 finalists. In particular, we looked at: score distributions by role; dungeon level exploration frequencies; most common causes of death; and other statistics like maximum armor class change, or episode length. More details can be found in Appendix B.

We found that the performance of all agents was significantly tied to their starting role and that such performance was often very heavy-tailed (see Figure 5 in Appendix B). For instance, "easy" starting roles like *Barbarian*, *Monk*, *Samurai* and *Valkyrie* would often have a higher median score than "harder" roles like *Healer*, *Rogue*, *Tourist* and *Wizard*, and also a much higher top score.

Since submissions were heavily incentivised to focus on their below-average characters, we noticed many strategies catered to this directly. We sometimes saw behaviours such as early termination of episodes beyond a certain internal score so as not to run down the clock (see Figure 7 in Appendix B), or the intentional restriction of the agent to the top-level dungeons (see Figure 6 in Appendix B). While the latter behaviour protects weak characters from dangerous monsters, it also likely contributes to the very high incidence of death due to "fainted of starvation" (see Figure 8 in Appendix B) since agents can run out of food easily if they do not explore the dungeon.

Although the winning agent set a new record for `NLE`, achieving a median score of 5,300, this is considered only just above a Beginner[8] score. This low result can be explained by the serious challenge posed by playing with difficult roles; outliers in the "easy" characters performed orders of magnitude better. For instance, 1 in 20 of the winning agent's *Valkyries* would get a score greater than 30,000, descend to dungeon level 10, and get to experience level 10 (typically not all in the same run). However, this is still far short of ascension, which requires the completion of a quest, around 50 dungeon levels, and the accumulation of various special items. This gap demonstrates the difficulty of designing agents that can deal with all the dangers in the complex world of NetHack and win the game.

8. A "Beginner" score is defined by the NetHack Wiki to be less than 2,000, or 1,000 for a *Wizard*, according to https://nethackwiki.com/wiki/Beginner

3.3. Symbolic vs. Neural Approaches

The NetHack challenge was designed to incentivise competition between neural and symbolic agents. This year, symbolic methods won, with notable advantages in multiple areas.

Participants building symbolic bots found it easy to define 'strategy'-like subroutines, thereby incorporating their domain-knowledge of NetHack into their bots. These bots were often equipped with elaborate routines for deciding when to exercise certain behavior based on rich, human-legible representations of the game state. Such large subroutines were found in all the top symbolic entries, which often had explicit strategies in code like "Find Minetown". This strategic play is very useful in the complex world of NetHack. In contrast, neural agents struggled in the area since hierarchical RL is still an open research problem. It is hard for agents to discover 'strategy'-like behaviour patterns in environments with a large action space and sparse reward. Moreover, it is unclear how to best provide RL agents with human prior knowledge about such strategies, for example, information from the NetHack Wiki (like armour strength or monster abilities). Lastly, symbolic bots excelled at leveraging long-term relationships in the game to contextualize and refine their behavior. In contrast, learning long-term relationships and performing credit assignment are still open problems in deep RL.

4. Approaches

In the following section, we first present the baselines made available to competitors, and then a description of the best performing symbolic, hybrid and neural entries by the teams themselves.

4.1. Baselines

Two baselines were made available to competitors over the course of the competition, specifically for the neural track. These baselines formed the basis of the vast majority of submissions, which were all high-throughput model-free policy gradient methods. No baselines were prepared for the symbolic track, meaning that all such solutions were coded from scratch, often spanning to many thousands of lines of code.

4.1.1. TorchBeast Baseline

The TorchBeast baseline is a modified version of the method presented in the original NeurIPS 2020 paper (Küttler et al., 2020), which is an implementation of IMPALA (Espeholt et al., 2018) in the open-sourced RL framework TorchBeast (2019). This model uses the `glyphs`, `blstats`, and `message` observations as inputs. The `glyphs` are factorised and embedded before passing into a CNN, while the `message` is encoded with a character CNN and the `blstats` is encoded with a feedforward neural network. These are all encoded and passed to an LSTM, that produces baseline and policy heads, albeit in a restricted action space. The architecture is shown in Figure 2. This baseline was released with the starter-kit[9] along with versions of the model trained for 0.25B and 0.5B steps.

9. https://gitlab.aicrowd.com/nethack/neurips-2021-the-nethack-challenge

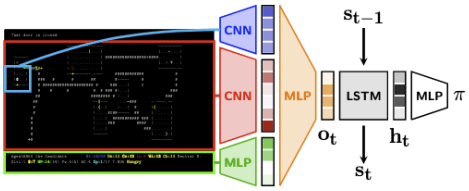

Figure 2: Neural network model used by the TorchBeast baseline

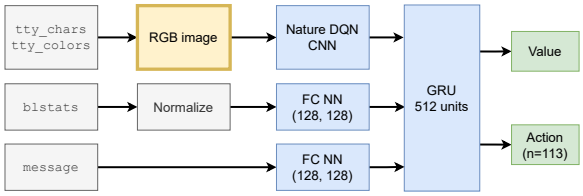

Figure 3: Neural network model used by the Sample Factory baseline.

4.1.2. Sample Factory Baseline

During the competition, the team 'Chaotic Dwarven GPT-5' open-sourced[10] a new baseline solution based on the Sample Factory (Petrenko et al., 2020) framework with asynchronous PPO and a RNN layer. This baseline is more performant than the TorchBeast baseline, reaching a mean score of approximately 700 (vs 400) with under 24 hours of training on a single RTX 2080Ti with 480 parallel `NLE` environments. This improved performance further lowered the barrier of entry for participants with weaker machines or fewer resources.

Instead of embedding the glyphs, this baseline renders the state as an RGB image, similar to what a human player sees on a cropped terminal screen. While this discards important details, like the specific monster types, the use of traditional convolutional layers speeds up training and allows agents to learn the associations of colors and shapes. Figure 3 details the network structure, which also uses `blstats` and the `message` observation.

One shortcoming of this baseline is the inability to learn the complex behaviour required by roles such as *Tourist*, *Healer* and *Wizard*, for which it achieves a low score. Another shortcoming is that the learning process is relatively slow as the agent is often stuck exploring different menus instead of making progress on the game itself.

4.2. Competitor Approaches

We now present the best symbolic, hybrid and neural approaches, described by the teams in their own words.

4.2.1. Team AutoAscend: Symbolic

There are two main parts to our framework:

Parsing and maintaining the knowledge about the game state. This consists of parsing ASCII pop-ups, remembering dungeon level state including non-current levels, using commands like `#terrain` to check covered objects by items and seen traps, keeping track of monsters including peaceful monsters, inventory and contents of the hero's bag, etc.

10. See https://github.com/Miffyli/nle-sample-factory-baseline, maintained by co-authors Anssi Kanervisto and Karolis Ramanauskas.

Decision making. We implemented a custom variation of a behaviour tree capable of dynamic tree expansion, including recursions. Strategies are selected based on the current game state.

`NLE` actions correspond to single keystrokes and therefore their behaviour depends on the context. For instance, to throw a projectile one usually needs to press 3 keys: '`t`' to initiate the throw action, a letter representing an item in the inventory, and a letter representing a direction. We implemented an additional level of abstraction to wrap many of such actions to make the solution more robust and easier to develop.

We introduced the abstraction of a *strategy* which comprises logic that handles a specific behaviour and usually consists of multiple actions, e.g., move to a given (x, y) position, or solve a Sokoban map. The idea behind strategies is their non-atomic nature and that they may be interrupted under specific conditions, e.g., the 'level exploration' strategy should be interrupted when noticing a monster in favour of the 'combat' strategy, which in turn should be interrupted by the 'emergency healing' strategy. Throughout the competition, we developed multiple strategies together with the conditions of when they should activate.

One example of a more complex strategy in our solution is combat, which is priority based. We defined a simplified action space with $4 \times 8 + 2 = 34$ actions: *movement*, *melee attack*, *ranged attack*, and *ray wand zap* – all in 8 directions; *write "Elbereth"*, and *wait*. We implemented a heuristic scoring function that assigns a score to every possible action; the one with the highest priority is subsequently executed. As a simplification, we always use the best available melee weapon, projectile, launcher, etc. taking into account properties like damage, to-hit bonus and the hero's weapon proficiency. For future work, there is huge potential in improving the action scoring algorithm with more advanced methods, such as an RL model trained on this simplified action space.

Another example of a strategy that resulted in a significant improvement, especially for non-combat oriented roles, is nutrition management. We use the following three rules which we found efficient: (1) eat fresh and safe-to-eat corpses from the ground until "satiated"; (2) if there are no such corpses, eat food rations gathered during exploration from the inventory in case the hunger state is "hungry" or worse; (3) as a last resort, pray when the hunger state is "fainting", but no more often than 500 turns after the previous prayer.

A more detailed list of strategies, behaviours, and features is presented in Appendix C, along with an image of the custom visualization and debugging tool we developed.

4.2.2. Team RAPH: Hybrid

We decided to apply a hierarchical approach to the challenge and construct an agent's policy from basic skills dedicated to solving concrete tasks: eating, fighting, dungeon exploration, and inventory management. The approach closely resembles the options framework of Sutton et al. (1999), in which a higher-level policy orchestrates the execution of eligible lower-level options until termination. This modularity allows us to build a hybrid neural-algorithmic method, where some skills can be trained, and others – hard-coded. In our case, we opted for a simplified higher-level policy and implemented it as a rule-based algorithmic decision system, that executes a lower-level skill on a first-fit basis. The priority and triggers of each skill were designed manually and determined based on our expert knowledge of rogue-likes and essential game AI.

One of the most important and complex skills, which accounts for the bulk of in-game score, is battling monsters. Fighting and combat require a complex policy to assess the surrounding topology of the level and choose an appropriate action: approach, outflank, avoid getting surrounded, decide on a melee or ranged attack, heal, wait or flee. To this end we train a deep neural RL agent based on the TorchBeast baseline, provided in Küttler et al. (2020), to learn a policy for the lower-level fighting skill. The agent has a discrete action space containing *eight* actions for directional movement or melee attacks, another *eight* actions for directional ranged attacks, and *three* actions for hard-coded composite controls such as waiting, praying and engraving "Elbereth", the latter warding off low-level aggressive monsters. The neural policy uses hand-crafted features related to the map, monster, and hero's vitals, extracted by the lower level algorithmic dungeon level mapping subsystem of the agent. We implement other skills as hard-coded algorithmic policies based on graph navigation algorithms and expert knowledge.

To train the agent we construct episodes by *pasting contiguous fragments* with transitions in which there is a *hostile monster within the field-of-view* of the agent. In other words, the steps performed by all other skills are "fast-forwarded", which makes the environment a partially observed semi-Markov decision process from the point of view of the agent, (Sutton et al., 1999). The inference of our RAPH agent works as presented in the Appendix, in algorithm 1. If necessary, we first handle pending NetHack's GUI events, such as responding to multi-part message logs. Next, we parse and update the dungeon representation and extract the features for the neural agent. Finally, we sample actions from either the RL agent or the hard-coded skill, whichever one currently holds control, taking into account the distance to the nearest monster.

Analysis of a trained policy shows that move or melee attack actions are used in 66% of steps, range attack in 21.6%, wait in 11%, "Elbereth" in 0.3%, and pray in 0.1%.

4.2.3. Team Kakao Brain: Neural

Our method is based on the V-trace actor-critic framework, IMPALA (Espeholt et al., 2018), which we used to optimize a policy network through end-to-end RL training from scratch. Here, we briefly describe our approach from the perspectives of 1) observation encoding, 2) separated action spaces, 3) network structure and 4) role-specific training. We provide more details in Appendix E.

First, we encode `messages`, `glyphs`, and an extended `blstats` as input observations. We also encode the information of *usable items*, *pickable items*, and *spells*, summarized in Figure 4, so we can explicitly utilize items and spells. Second, we separate the action space into *action-type*, *direction*, *use-item*, *pick-item*, and *use-spell*. In the original action space, a single action can be used for different meanings according to the game status. For example, key 'a' (action 24) is used for applying tools as well as for selecting an item or a spell. Our separated and somewhat hierarchical action space clarifies the meaning of each action and also reduces the whole space to the valid action space at each time.

Our network structure is depicted in Figure 11 in Appendix E. The encoded `glyphs` are fed to the CNN while the other observations including tokenized `messages` and extended `blstats` are fed to the MLPs. These features as well as the last action type are then concatenated into the GRU network (Cho et al., 2014) for exploiting past information. Moreover, we use Transformer (Parisotto et al., 2020) to combine the GRU output with the

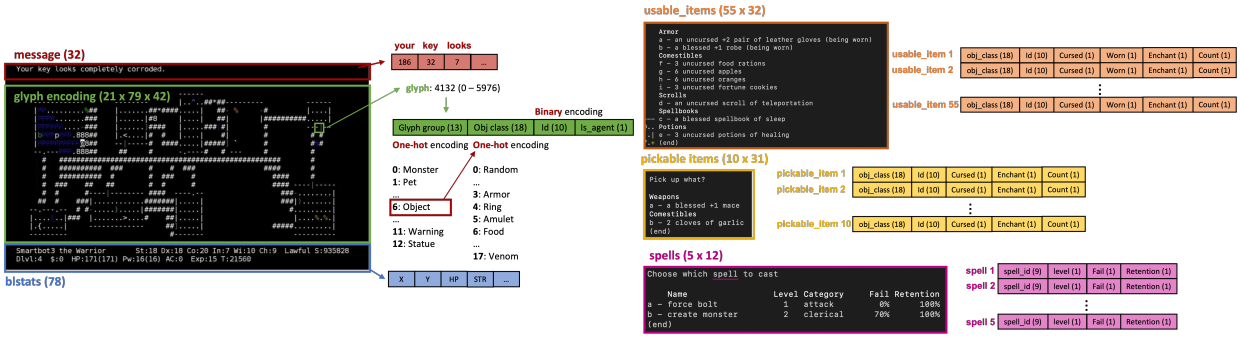

Figure 4: Observation Encoding of Team KakaoBrain: one-hot encoding for glyph group and object class, and binary encoding for id.

information of *usable items*, *pickable items*, and *spells*. The transformer output is then fed to the value head and the multiple policy heads according to the separated action spaces.

To encourage role-specific strategies, we make use of role-specific agents trained by role-specific reward shaping. We maintain such agents for *Healer*, *Ranger*, *Rogue*, *Tourist*, and *Wizard* while the universal agent is used for the other roles. Each policy trains for 10B steps on 4 nodes, with 224 CPU cores and 16 V100 GPUs in total. We modify Sample Factory (Petrenko et al., 2020) for a throughput of 100K frames per second.

5. Conclusion

The Challenge produced significant progress in NetHack, including a 5× improvement in deep RL approaches, and a new open-source baseline. It also provided a valuable point of comparison for neural (deep RL) and symbolic methods, with symbolic bots achieving 3× the median score of deep RL bots in the dungeon. Notably, the game remains unsolved. The best agents' median score is several orders of magnitude short of a typical (human) 'ascension'. As argued in Küttler et al. (2020), the `NLE` environment is at the frontier for RL research and it remains to be seen which methods will be able to scale to the point of reliably beating the game.

Future challenges should focus on refining evaluation metrics, such that the ranking of agents better reflects progress in the game. This year many teams indulged in strategies that optimised in-game score at the expense of progressing into the dungeon for ascension. It is possible a new metric, perhaps involving the checkpointing of achievements (as is done for Junethack[11]) may be more successful in steering agents to ascension. Future challenges may also seek to proactively encourage teams to exploit external knowledge (e.g., from the NetHack Wiki, or possibly 'source-diving' into NetHack's C code), learn from recordings of human play (e.g., `ttyrec` game replays from alt.org), or test skill acquisition (e.g., using MiniHack (2021)). These are all areas of research that fit naturally into NetHack's ecosystem and pose challenges that would benefit from open competitive benchmarking.

11. https://nethackwiki.com/wiki/Junethack/FAQ#Achievements_.2F_Trophies

Acknowledgments

The organisers would like to thank Meta AI, DeepMind and AIcrowd for their generous sponsorship of this project.

References

Jan Krajicek. BotHack. https://github.com/krajj7/BotHack, 2014. Accessed: 2022-02-04.

Aransentin Breggan Hampe Pellsson. SWAGGINZZZ. https://pellsson.github.io/, 2019. Accessed: 2020-05-30.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *CoRR*, abs/1606.01540, 2016.

Kyunghyun Cho, B van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In *Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST-8), 2014*, 2014.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416. PMLR, 2018.

Eva Myers. List of Nethack spoilers. https://sites.google.com/view/evasroguelikegamessite/list-of-nethack-spoilers, 2020. Accessed: 2020-06-03.

Kenneth Lorber. NetHack Home Page. https://nethack.org, 2020. Accessed: 2020-05-30.

Heinrich Küttler, Nantas Nardelli, Thibaut Lavril, Marco Selvatici, Viswanath Sivakumar, Tim Rocktäschel, and Edward Grefenstette. TorchBeast: A PyTorch Platform for Distributed RL. *arXiv preprint arXiv:1910.03552*, 2019. URL https://github.com/facebookresearch/torchbeast.

Heinrich Küttler, Nantas Nardelli, Alexander H. Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The NetHack Learning Environment. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

NetHack Wiki. NetHackWiki. https://nethackwiki.com/, 2020. Accessed: 2020-02-01.

Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. Stabilizing transformers for reinforcement learning. In *International Conference on Machine Learning*, pages 7487–7498. PMLR, 2020.

Aleksei Petrenko, Zhehui Huang, Tushar Kumar, Gaurav Sukhatme, and Vladlen Koltun. Sample factory: Egocentric 3d control from pixels at 100000 fps with asynchronous reinforcement learning. In *ICML*, 2020.

Eric S. Raymond. *A Guide to the Mazes of Menace*, 1987.

Mikayel Samvelyan, Robert Kirk, Vitaly Kurin, Jack Parker-Holder, Minqi Jiang, Eric Hambro, Fabio Petroni, Heinrich Küttler, Edward Grefenstette, and Tim Rocktäschel. Minihack the planet: A sandbox for open-ended reinforcement learning research. *CoRR*, abs/2109.13202, 2021. URL https://arxiv.org/abs/2109.13202.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112 (1-2):181–211, August 1999. doi: 10.1016/s0004-3702(99)00052-1. URL https://doi.org/10.1016/s0004-3702(99)00052-1.

TAEB. TAEB Documentation: Other Bots. https://taeb.github.io/bots.html, 2015. Accessed: 2020-01-19.