Advanced Combat System for MiniHack - Project Report

Giuseppe Di Palma, Daniel Wahle

22.01.2025

1 Introduction

MiniHack [3] is a research platform based on NetHack, an open-source single-player roguelike video game. Roguelike games are turn-based dungeon crawlers with procedurally generated levels and permadeath. In NetHack, players navigate through dungeons and face a wide variety of challenges, such as monsters, traps, and puzzles. There are many possible actions and outcomes at any given moment, which makes the game complex. MiniHack simplifies and modularizes this complexity and provides environments designed to test specific capabilities of AI agents. For example, an environment might focus on pathfinding, resource management, or combat tactics.
This project aims to build a MiniHack agent that can handle multiple monsters of different types at the same time. Two approaches are explored: the first uses a first-order logic knowledge base implemented in Prolog, while the second relies on reinforcement learning with TorchBeast.

2 Related Works

Prolog [1], a logic programming language rooted in first-order logic, offers powerful tools for reasoning, planning, and decision-making, which makes it a good fit for gaming applications. Alongside knowledge-based approaches, reinforcement learning (RL) has gained significant traction for its ability to train agents through trial-and-error interactions. TorchBeast [2], a popular RL framework built on PyTorch, is widely used for scaling RL experiments, including those conducted in environments like MiniHack. TorchBeast leverages high-performance distributed training to optimize policies based on raw state-action pairs, enabling agents to learn complex behaviors without predefined rules.

3 Methodologies

The environment in which the agent is situated is a square map of a simple nature, with walls erected along the perimeter, within which the enemies, the potion and the agent are randomly positioned.
For the FOL-based agent, the knowledge base describes all the rules necessary to perform an action, and a Python program is used to interact with the MiniHack game based on the decisions of the knowledge base.
The following rules define what actions the agent can take based on the current state of the game.

- `action(wield(Key))`: If the nearest enemy cannot be defeated with the current weapon, the agent will switch to a weapon it has that can defeat the enemy.
- `action(shoot(Direction))`: If an enemy is within the range of the agent's weapon, the agent will shoot in the appropriate direction.
- `action(attack(Direction))`: If a nearby enemy can be defeated with the current weapon and the agent is healthy, the agent will attack.
- `action(move_towards_enemy(D))`: The agent moves toward a beatable enemy if the agent is healthy.
- `action(run(OppositeDirection))`: The agent runs away if it is unhealthy or cannot beat a nearby enemy.
- `action(drink(Key))`: The agent drinks a health potion if it is unhealthy.
- `action(pick)`: If the agent is stepping on a pickable item (e.g., a weapon or potion), it picks it up.
- `action(move_towards_potion(Direction))`: If the agent is unhealthy, it moves toward a health potion in a safe direction.

As stated in Section 1, the TorchBeast tool facilitates efficient and effective Reinforcement Learning training. To train the neural network in the same environment as the FOL-based agent, a custom mini hack environment has been registered. Furthermore, the neural network used in the tests is a Convolutional Neural Network, the baseline model available in TorchBeast.

4 Assessment

We evaluated the agent over a total of 100 episodes, with each episode allowing a maximum of 200 steps. The agent's objective was to defeat all the monsters on the level. The reward structure incentivized the agent for successfully defeating monsters, with each kill yielding a reward of 1.

We performed five tests.

- **Test 1 - Non-Flying Monsters:** The agent must defeat two non-flying monsters: a `kobold` and a `goblin`.
- **Test 2 - Flying Monsters:** The agent must defeat two flying monsters: a `bat` and a `giant bat`.
- **Test 3 - Two Mixed Monsters:** The agent must defeat two monsters: a `kobold` and a `giant bat`.
- **Test 4 - Three Mixed Monsters:** The agent must defeat three monsters: a `kobold`, a `goblin`, and a `giant bat`.
- **Test 5 - Three Mixed Monsters:** The agent must defeat three monsters: a `kobold`, a `giant bat`, and a `bat`.

And for each of these, we performed three different runs.
Table 1 shows the results for the FOL-based agent, and Table 2 shows the results for the TorchBeast-based agent. The mean values are bolded, and variances are in regular text.

Test	**Mean return**[1]	**Success rate (%)**	**Mean SPE**[2]	**Mean SPWE**[3]
1	**1.85** $\pm$ 0.03	**92.33** $\pm$ 1.24	**6.68** $\pm$ 0.18	**10.97** $\pm$ 0.41
2	**1.79** $\pm$ 0.02	**87.66** $\pm$ 2.62	**5.07** $\pm$ 0.28	**8.15** $\pm$ 0.43
3	**1.86** $\pm$ 0.07	**92.33** $\pm$ 3.85	**6.59** $\pm$ 0.13	**10.83** $\pm$ 0.05
4	**2.58** $\pm$ 0.06	**78.66** $\pm$ 3.40	**8.82** $\pm$ 0.12	**15.31** $\pm$ 0.20
5	**2.40** $\pm$ 0.07	**70.00** $\pm$ 2.82	**7.80** $\pm$ 0.67	**13.49** $\pm$ 0.48

Table 1: Test results for the FOL-based agent.

Test	**Mean Return**	**Mean SPE**
1	**0.51** $\pm$ 0.39	**84.26** $\pm$ 4.89
2	**0.37** $\pm$ 0.10	**80.02** $\pm$ 3.00
3	**0.77** $\pm$ 0.06	**62.47** $\pm$ 17.96
4	**0.86** $\pm$ 0.36	**82.27** $\pm$ 7.12
5	**0.84** $\pm$ 0.20	**74.12** $\pm$ 8.13

Table 2: Test results for the TorchBeast-based agent.

[1] The mean number of points attained by the agent across all episodes.
[2] This is the mean number of steps per episode performed by the agent across the three runs.
[3] This is the mean number of steps performed by the agent across three runs, but only for those runs where the episode was won.

The mean return for the FOL-based agent and the TorchBeast-based agent is different because of the way in which the points are allocated. In the FOL implementation, the agent acquires one point for each monster that is killed. By contrast, in the TorchBeast implementation, the agent receives a point if it is able to win the episode, and zero points otherwise. In the TorchBeast case, this can be regarded as a success rate for the FOL agent.
The findings demonstrate that the FOL implementation yielded superior performance in comparison to the TorchBeast implementation. This observation can be substantiated by the variance of the `mean return` metrics, wherein the TorchBeast implementation exhibited a higher variance despite employing a more limited value scale. Furthermore, the number of steps exhibited a similar trend, with the FOL-based agent demonstrating superior performance.
However, it is crucial to emphasise that the TorchBeast-based agent does not necessitate any prior knowledge of the environment in which it is to be trained, thus rendering it a highly flexible approach. Furthermore, despite the considerable variance in performance, this approach is capable of attaining optimal performance levels. With meticulous fine-tuning and an adequate allocation of training time, it is likely to surpass the FOL-based agent, which requires extensive prior knowledge of the environment.

5 Conclusion

In this project, two approaches for developing an advanced combat agent in the MiniHack environment were successfully implemented and evaluated: a Prolog-based first-order logic (FOL) agent and a reinforcement learning (RL) agent using TorchBeast. The results demonstrated that the FOL-based agent consistently outperformed the RL-based agent in terms of mean return and efficiency, owing to its reliance on extensive prior knowledge. Conversely, the RL-based agent exhibited greater flexibility and the potential for achieving optimal performance with further tuning and training. This work underscores the trade-offs between knowledge-based systems and data-driven approaches in AI applications for complex gaming environments. Future work could explore hybrid methods to combine the strengths of both approaches for enhanced performance.

References

[1] Alain Colmerauer and Philippe Roussel. *The birth of Prolog*, page 331–367. Association for Computing Machinery, New York, NY, USA, 1996.

[2] Heinrich Küttler, Nantas Nardelli, Thibaut Lavril, Marco Selvatici, Viswanath Sivakumar, Tim Rocktäschel, and Edward Grefenstette. Torchbeast: A pytorch platform for distributed rl, 2019.

[3] 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, 2021.

Appendix

A Team contributions

- **Giuseppe** focused on both MiniHack and TorchBeast environment setup. He worked on the Prolog-based knowledge base and the TorchBeast agent. He also contributed to testing.
- **Daniel** worked on the Prolog-based knowledge base and was responsible for the documentation. He also contributed to testing.

B GitHub Metrics

The GitHub repository with the metrics can be found here.

C Relationship with the Course

This project relates to several AI concepts from the course. The first approach explored in this project uses first-order logic, which relates to the course's coverage of first-order logic and inference in first-order logic. The second approach uses reinforcement learning. Although reinforcement learning was not covered in depth in the course, it was mentioned several times.