MiniHack

Francesco Romeo, Raffaele Cadau

July 10, 2025

Introduction

This report describes a project aimed at developing an agent for the *MiniHack* research environment, a platform built on the roguelike video game *NetHack*.

The approach explored in this work is based on a knowledge base implemented using first-order logic, written in the *Prolog* programming language. This choice stems from the need to model complex behaviors through robust and declarative logical inference. Thanks to this knowledge structure, the agent was able to perform targeted strategic decisions, considering not only the enemies' attributes but also the environmental context and its own state.

The results obtained show that the first-order logic-based agent exhibited high performance in terms of average return and efficiency. This can be attributed to its extensive prior knowledge of the environment, which allowed for fast and rational responses to in-game events.

Related Work

Prolog[1], a logic programming language rooted in first-order logic, offers powerful tools for reasoning, planning, and decision-making. Its declarative nature and inference capabilities make it particularly well-suited for applications in games where explicit logic and predefined rules can serve as the backbone for agent behavior. The strength of Prolog lies in its ability to define a structured and interpretable set of rules, enabling agents to act based on a well-defined understanding of the environment rather than probabilistic generalizations.

Technical Approach (Methodologies)

The design and implementation of the MiniHack agent were grounded in a methodology based on First-Order Logic (FOL), aiming to enable symbolic reasoning in a dynamic and partially observable environment.

The environment in which the agent operates is a simple square grid, bounded by walls. Since the environment changes dynamically — with monsters and stairs spawning in random locations — performance variations may be attributed either to the intrinsic differences between the agent algorithms or to the stochastic variability of the environment itself.

Knowledge Base Design and Reasoning Structure

Several key predicates govern the behavior of the agent:

- `agent_pos(X,Y)`: Represents the agent's current position on the map.
- `goal(X, Y, Z)`: Encodes the target symbol the agent must reach or interact with.
- `walkable(X,Y)`: Defines whether a given position is traversable.
- `has(X, Y, Z, W, K, J)`: Represents inventory management, including weapons, potions.
- `health/1` and `health_max/1`: Define the agent's current and maximum health, used to evaluate whether the agent is "healthy".
- `is_monster(X, Y)` and `maybe_monster(X, Y)`: Encode prior beliefs or dynamic inference about symbols representing enemies and their threat level.

One of the strengths of this system is the combination of symbolic proximity (`near_goal/1`), health thresholds (`healthy/0`), and inferred threat estimation (`beliefSeeMonster/4`). The latter predicate, in particular, enables dynamic reasoning about hostile entities: if the agent encounters a symbol it does not recognize and subsequently loses HP, the system infers it might be a dangerous monster. Additionally, a structured representation of possible actions is maintained. Examples include:

- `use(Key, 'q')`: Logic for quaffing potions based on health level.
- `use(Key, 'w')`, `use(Key, 'W')`, `use(Key, 'T')`: Executes an equip action based on the current state of the inventory and equipped gear. The agent attempts to select and equip the best available weapon — typically prioritizing the most effective weapon — in order to optimize combat performance.

The KB is designed to be dynamically extensible, using the `:- dynamic` directive for all key predicates, enabling learning and adaptation across episodes.

This rule-based architecture allows for **transparent, interpretable behavior**. Unlike black-box models, each action is the direct consequence of verifiable logic clauses. The modularity of Prolog facilitates the addition of new rules, enabling future expansion.

Results and Evaluation

The performance of the proposed logic-based agent was rigorously assessed through a structured series of experiments conducted within the MiniHack environment. The evaluation comprised **30 independent episodes** for file, each capped at a maximum duration of **200 discrete time steps**. This setup allowed for a comprehensive analysis of the agent's strategic behavior and its ability to adapt across multiple, varied scenarios.

Evaluation Metrics

To quantify the agent's performance, two principal metrics were defined:

- **Success Rate (%)**: This metric represents the percentage of episodes in which the agent successfully win. It serves as a direct measure of task effectiveness.
- **Average Steps Per Episode (ASPE)**: This captures the average number of steps taken by the agent per episode, independent

of outcome. It reflects the agent's efficiency in achieving its objectives—lower values are indicative of faster and more decisive strategies.

These two metrics together offer a balanced perspective: the success rate emphasizes the agent's effectiveness, while the ASPE highlights operational efficiency. An ideal agent would simultaneously maximize the former and minimize the latter, though this trade-off is often nontrivial in practice.

Quantitative Analysis

Table 1 reports the detailed performance statistics across five distinct level configurations, each characterized by varying monster types and spatial complexities.

Level File	**Victories**	**Episodes**	**Avg. Steps**
`level/monster_0.des`	26	30	5.57
`level/monster_d.des`	18	30	41.97
`level/monster_f.des`	8	30	26.37
`level/monster_fo.des`	19	30	6.97
`level/monster_o.des`	26	30	48.53

Table 1: Performance statistics for different level configurations: number of victories, total episodes, and average number of steps per episode.

Several trends emerge from this dataset. The agent performs exceptionally well on levels such as `monster_0.des` and `monster_o.des`, where it achieves a high success rate (26 out of 30) but with dramatically different average step counts (5.57 vs. 48.53), indicating a significant variance in environmental difficulty or combat complexity. On the contrary, levels like `monster_f.des` posed considerable challenges: the agent succeeded in only 8 episodes, despite requiring fewer steps on average than some higher-performing levels. This suggests a possible mismatch between the agent's reasoning capabilities and the specific behaviors or interactions imposed by that monster configuration.

Overall, the agent demonstrates robust performance under certain conditions but reveals limitations in more complex or less predictable contexts, highlighting areas where further refinement may be necessary.

Conclusion

This project successfully introduced and empirically validated a logic-based combat agent designed for the MiniHack environment. Leveraging First-Order Logic (FOL) and implemented using the Prolog programming language, the agent exhibited competent tactical behavior and was able to systematically reason about the environment and its adversaries.

References

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

Appendix

A. Team Contributions

The project was developed through a highly collaborative effort between the two contributors.

- **Francesco Romeo** and **Raffaele Cadau** collaboratively designed the logic-based agent's architecture, jointly implemented the Prolog modules, and coordinated the experimental setup within the MiniHack environment.
- Both contributors participated equally in data analysis, performance evaluation, and the preparation of all written materials, ensuring a unified and coherent presentation of the project.

B. GitHub Repository and Development Metrics

The complete source code, along with detailed development metrics and documentation, is publicly accessible via the project's GitHub repository. You can find it here.

C. Relationship to the Course

This project is directly connected to various core topics in Artificial Intelligence discussed throughout the academic course. The approach explored—centered on First-Order Logic—aligns with the course content covering logical inference, rule-based systems, and symbolic reasoning. This practical application offered an opportunity to deploy and assess a fundamental AI paradigm in a realistic and complex environment, reinforcing the theoretical principles introduced during lectures.