Solving Scaleable POMDP Mazes with Parallel Temporally Extended Task Sequences

Andy Edmondson[1,2], **Ronald P.A. Petrick**[1]

[1]School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, UK,
[2]School of Informatics, The University of Edinburgh, Edinburgh, UK,
Andy.Edmondson@ed.ac.uk, R.Petrick@hw.ac.uk

Abstract

Search and Rescue domains may involve completing initially unknown numbers of temporally extended task sequences within partially observable non-stationary environments. Many learning approaches struggle in such environments because doing so requires memory of value transitions and previously visited states. While POMDP environments with short sequences of subtasks exist, they do not require completion of multiple sequences in parallel and do not require a policy to cope with initially unknown numbers of parallel tasks, variable subtask sequencing and non-stationarities. This work presents a new gridworld maze environment which requires agents to remember subtask progress across an unknown and potentially large number of parallel subtasks while exploring an unknown environment and rescuing an unknown number of non-stationary people. Scaling to 1000 rooms with 1500 people, this environment can aid researchers evaluate solutions to this class of problem. The demonstration shows a Priority-Based Reward-Machine Switching (PBRMS) agent, a novel approach which decouples Reward Machine states from task sequence instances, fully exploring a 100 room environment with 500 people across 8962 timesteps.

Code — https://github.com/Levinin/rescue_gridworld_env
Video — https://youtu.be/851cbv87U3w

Introduction

Search and Rescue (SAR) is a complex domain and may involve the completion of initially unknown numbers of temporally extended task sequences within partially observable non-stationary environments. This combination of partial observability and non-stationarity produces subtask relationships that are: **inherently contingent**, since subtasks become available only when their preconditions are observed, and non-stationarities mean these may become true or false independently of the agent; **interruptible**, such as when a person is observed as the agent approaches a door; **emergent**, since the available subtask structure at a given point depends on discovered environmental context. Additionally, environments are **indeterminately large**, since there may be any finite number of tasks which must be tracked independently and in parallel.

Where the agent receives partial observations o_t which do not uniquely identify the underlying state or encode transition histories, approaches such as Reward Machines (RMs) and Hierarchical RMs provide subtask decomposition and encode the required memory (Toro Icarte et al. 2022; Furelos-Blanco et al. 2023). LTL-based composition approaches similarly use automaton states to guide task execution (Bergeron, Serlin, and Leahy 2024), while Reinforcement Learning (RL) approaches often try LSTMs (Bakker 2001). This memory theoretically renders otherwise non-Markovian problems Markovian over the joint observation-automaton state space, enabling Reinforcement Learning methods to learn effectively.

However, in SAR domains it is always possible an agent will discover parallel subtask sequence $k > n$ when only $\mathcal{M} = [M_1, M_2, \ldots, M_n]$ RMs were specified, and LSTM approaches scale poorly with task sequence length. This motivates the need for an environment which requires planners and learning agents to scale well and cope with the indeterminately large number of potential tasks, variable sequencing and non-stationarities found in SAR environments.

The demonstration showcases a new Gymnasium (Towers et al. 2025) environment, available on GitHub and PyPI, incorporating these characteristics and being solved by a novel Priority-Based Reward-Machine Switching (PBRMS) agent (Edmondson and Petrick 2026), an RM-based approach which decouples RM states from task sequence instances, allowing parallel execution of an arbitrary number of sequences while contingently switching according to task priorities and subtask sequence progress.

Environment Design

The grid-world shown in Figure 1 was created as a Gymnasium environment (Towers et al. 2025) in Python. It is designed such that the agent must complete sequential tasks while adapting to partial observations which reveal parallel and higher priority tasks. Procedurally generated on each reset, the entire layout including room placement, numbers of task-relevant items and their locations is randomised to present a different number and sequence of subtasks each episode. The agent starts in one room, and must reach an exit located in another.

This design captures the key challenges identified above of contingent actions, interruptibility, emergence and non-

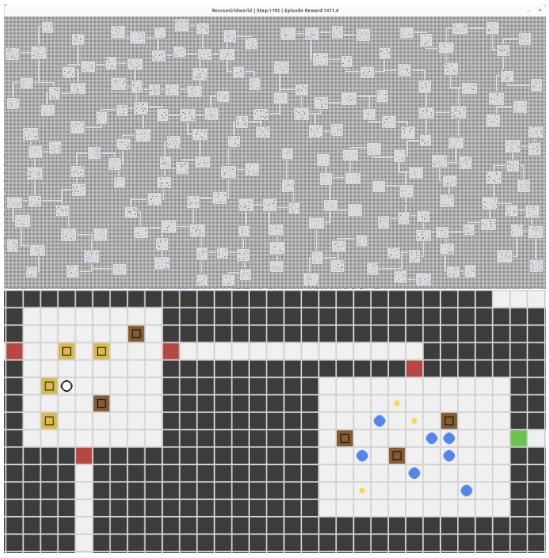

Figure 1: Example 350 room, 1500 people layout with standard rendering (top) and closeup demo render (bottom).

stationarity. Key is that, because the number and arrangement of subtasks vary, it evaluates whether an agent can support that class of domain whereby differing numbers of parallel subtask instances must be completed. Successful operation across these procedurally generated layouts therefore demonstrates the ability to manage varying and large numbers of subtask instances without redesign or retraining.

During generation, *people* (blue dots) are placed randomly throughout the rooms. The agent is expected to *talk* to those it discovers, who then follow the agent, sharing the same square. During the episode, people not yet rescued move to a random adjacent empty square every 3 timesteps (to avoid a slow chase across the map) making the environment non-stationary. To make its way between rooms, the agent must *collect keys* from the floor (yellow dots), *unlock cupboards* containing keycards (brown squares with black markings), *collect keycards* from the cupboards, and use these keycards to *unlock doors* (red squares). Each key only opens one cupboard, each keycard one door, and it is not enough to collect all keys then unlock all cupboards (and so on) as some keys and keycards are only available after unlocking subsequent doors. Some doors and cupboards may also initialise as unlocked, so the sequence of actions required to move from room to room is not always the same. To ensure solvability, the environment validates its procedural generation by creating a plan from start to exit, restarting generation if no plan is found. The plan path can be displayed on the environment rendering.

The environment's observation is a 2D $n \times n$ (where n is any odd number) $360°$ line-of-sight (LoS) window, centred on the agent (see Figure 2). LoS is determined using a double pass Bresenham's line algorithm with pinch checks to ensure the agent only sees unobscured locations. Perfect object detection of observed items is assumed, meaning the agent can (for instance) identify the door number for each

```
[[  5   0   0   0   0   0   0]
 [  5   0   0   0   0  60   0]
 [  5  70   0   0   0   0   0]
 [ 10   0   0   0   0   0   0]
 [  5   0   0   0  30   0   0]
 [  5   0   0   0  30 100 100]
 [  5  70   0   0 100 100 100]]
```

Figure 2: Example partial observation window. The agent is at the centre, code 100 shows occlusions by code 30.

visible keycard, this provided as part of the observation. The tile codes are as follows: 0 = empty, 5 = wall, 10 = locked door, 20 = unlocked door, 30 = locked cupboard, 40 = unlocked cupboard, 50 = unlocked cupboard with keycard present, 60 = key present, 70 = person present, 80 = exit, 100 = occluded and 255 = unknown. Values are spread through the 0-255 range to give contrast to CNN feature extractors. The agent interacts with the environment through a set of 9 action codes: 0 = up, 1 = down, 2 = left, 3 = right, 4 = collect key, 5 = unlock cupboard, 6 = collect keycard, 7 = unlock door and 8 = talk to person.

A cost of -0.01 is applied each timestep, with a reward of 5 for completion of actions indexed 4 to 8. Exiting attracts a reward of 5, and exiting having rescued everyone an additional 50. Since each key only opens one cupboard and each keycard one door, the sequences $key_i \rightarrow cupboard_i \rightarrow keycard_i \rightarrow door_i$ must be tracked by the agent while it explores sufficiently to rescue everyone.

PBRMS Results and Conclusion

Evaluation with the novel PBRMS agent (Edmondson and Petrick 2026), an RM-based approach which decouples RM states from task sequence instances, and can therefore solve configurations of arbitrary size without reconfiguration or retraining, shows that both environment and PBRMS scale wall. During evaluation the environment and PBRMS agent operated with configurations to 500 rooms and 500 people, 1000 locked doors, cupboards, keys and keycards rendering at 10 Hz. With 1000 rooms and 1500 people it renders at 3 Hz, and without rendering runs at 50 Hz.

To evaluate the difficulty of learning this environment, and since recurrent networks have often been tried within POMDP environments (Bakker 2001; Meng, Gorbet, and Kulić 2021; Li et al. 2024), an LSTM-PPO agent from SB3's Contrib library (Raffin et al. 2021) was used as a baseline. Using proven perception layers for this environment, it was trained for 2M steps with rewards for correctly-ordered subtask completion. This agent performed poorly, completing a 2-room configuration in few than 10% of cases.

The demonstration video shows the environment operating, a scalable PBRMS agent solving a 100 room configuration with 500 people in 8962 timesteps. While difficult for standard RM, HRM or DRL agents, it is solvable using methods appropriate to the challenges presented.

Compute was a HP ZBook with 128 GB RAM, Intel Core i9-13950HX CPU and Nvidia RTX 5000 Mobile Ada GPU.

References

Bakker, B. 2001. Reinforcement Learning with Long Short-Term Memory. In Dietterich, T.; Becker, S.; and Ghahramani, Z., eds., *NeurIPS*, volume 14. MIT Press.

Bergeron, T.; Serlin, Z.; and Leahy, K. 2024. CompLTL: Temporal Logic Planning via Zero-Shot Policy Composition. In *AAAI 2025 Workshop on Generalization in Planning (GenPlan)*. Philadelphia.

Edmondson, A.; and Petrick, R. P. A. 2026. PBRMS: Priority-Based Reward-Machine Switching for Context-Aware Reinforcement Learning in Non-Stationary POMDPs. In *ICAPS 2026 PRL Workshop, In Preparation*.

Furelos-Blanco, D.; Law, M.; Jonsson, A.; Broda, K.; and Russo, A. 2023. Hierarchies of Reward Machines. In *ICML*, volume 40. Honolulu, Hawaii: PMLR.

Li, A. C.; Chen, Z.; Klassen, T. Q.; Vaezipoor, P.; Icarte, R. T.; and McIlraith, S. A. 2024. Reward Machines for Deep RL in Noisy and Uncertain Environments. In Globerson, A.; Mackey, L.; Belgrave, D.; Fan, A.; Paquet, U.; Tomczak, J.; and Zhang, C., eds., *NeurIPS*, volume 37, 110341–110368. Curran Associates, Inc.

Meng, L.; Gorbet, R.; and Kulić, D. 2021. Memory-based Deep Reinforcement Learning for POMDPs. In *2021 IEEE/RSJ IROS*, 5619–5626.

Raffin, A.; Hill, A.; Gleave, A.; Kanervisto, A.; Ernestus, M.; and Dormann, N. 2021. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *JMLR*, 22(268): 1–8.

Toro Icarte, R.; Klassen, T. Q.; Valenzano, R.; and McIlraith, S. A. 2022. Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning. *JAIR*, 73: 173–208.

Towers, M.; Kwiatkowski, A.; Terry, J.; Balis, J. U.; Cola, G. D.; Deleu, T.; Goulão, M.; Kallinteris, A.; Krimmel, M.; KG, A.; Perez-Vicente, R.; Pierré, A.; Schulhoff, S.; Tai, J. J.; Tan, H.; and Younis, O. G. 2025. Gymnasium: A Standard Interface for Reinforcement Learning Environments. arXiv:2407.17032.