B

Reinforcement Leaning Agent

Reinforcement Learning (RL) focuses on agent-based decision-making to maximize cumulative rewards within a given environment. Unlike supervised learning methods where models are trained on labeled datasets, RL agents must learn through interaction, guided only by sparse reward signals [53]. As noted by A.G. Barto, RL relies on two foundational components: search and memory [54]. Search involves trial and error to explore possible actions, while memory allows the system to retain and reuse information about what has worked well in past situations. Given a well-defined environment and reward function, RL can be a powerful framework for developing agents capable of aligning with intended goals—even learning subtle behaviors that may not have been explicitly designed.

However, RL also presents several challenges. Compared to supervised learning, it typically demands significantly more computational resources. Additionally, once an agent is trained in a specific environment, its learned policy often struggles to generalize to new or slightly altered environments.

To train RL agent on Level 1, the level was converted into a **Gymnasium** environment. Gymnasium is the actively maintained fork of OpenAI's Gym— a Python library for developing and testing RL algorithms [55], [56]. To create a custom Gym environment, the designer must configure the **observation** and **action** spaces, which define the possible inputs (actions taken by the agent) and outputs (observations returned by the environment) [56]. These spaces can be either discrete or continuous, depending on the task. In addition to these definitions, two core functions are required: the `reset` function, which initializes a new episode, and the `step` function, which

updates the environment state based on the agent's action. The **`step`** function must compute the next observation, the reward signal, and whether the episode has ended. With these components in place, the Level 1 environment was implemented as a custom class, `GridWorldEnv`, as described in Pseudocode 3.

Algorithm 3 `GridWorldEnv` Core Mechanisms

function INIT
    Define **action space** as 4 discrete actions (up, down, left, right)
    Define **observation space** as flattened grid of tile values
end function
function RESET
    Reset to initial grid configuration
    Reset step counter and collected chip set
    return initial observation
end function
function STEP(action)
    Move RL agent and evaluate resulting tile:

- Hazard → end episode with penalty
- Chip → collect and reward
- Socket → unlock if all chips collected and reward
- Exit → end episode with the biggest reward

    Apply small step penalty
    Move the other agent with sampled action
    return observation, reward, termination flags
end function
function _MOVE(position, action)
    Compute new coordinates from direction
    if within bounds and not wall **then**
        return new position
    else
        return original position
    end if
end function
function _GET_OBS
    Overlay agent positions on grid
    return flattened grid as observation
end function

After setting up the Gymnasium environment, the agent was trained using the Stable Baselines3 (SB3) library, which offers a collection of widely used reinforcement learning algorithms—including PPO, DQN, A2C, SAC, and TD3—implemented in PyTorch [57]. For this experiment, the agent was trained using the Proximal Policy Optimization (PPO) algorithm due to its performance across a wide range of RL tasks. Detailed parameter configuration is shown in the table B.1.

The experiment began by training an agent for 2 million timesteps, which served as a baseline model for subsequent reward function tuning. Building upon this base agent, additional training

Table B.1: PPO Training Hyperparameters

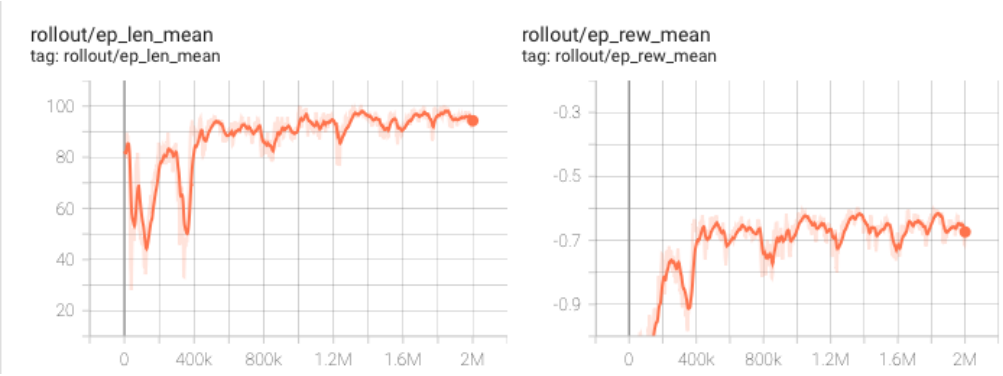
Parameter	**Value**
Policy	`MlpPolicy`
Learning Rate	3×10^{-4}
Discount Factor (γ)	0.99
GAE Lambda (λ)	0.95
Steps per Update (`n_steps`)	2048
Batch Size	64
Epochs per Update	10
Clipping Range	0.2
Random Seed	Fixed (None)

sessions were conducted for 1 million timesteps each, specifically targeting performance on Level 1. Despite a total of 3 million timesteps of training, the agent ultimately failed to learn how to complete the level. Figure B.1 illustrates the training trends for both the base agent and the subsequent runs.

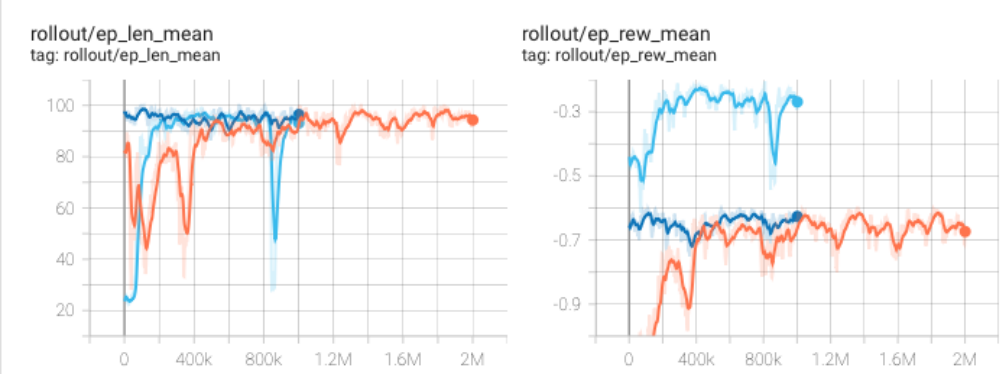
As shown in the figure, the agent's mean reward consistently remained on the negative side, indicating unsuccessful training. Notably, a significant behavioral difference can be observed between the runs shown in Figure B.1b and Figure B.1c. In Figure B.1b, the agent exhibits a higher mean reward and longer episode length, suggesting that it learned to navigate around hazardous tiles, though without completing the level. In contrast, Figure B.1c shows both lower mean reward and shorter episode lengths. This decline resulted from modifying the reward function to assign a larger negative penalty per step, aimed at discouraging inefficient behaviors such as aimless wandering or repeatedly visiting the same locations. However, this adjustment backfired—the agent began stepping into hazardous tiles more frequently in an apparent attempt to avoid the accumulating negative reward.

Finally, Figure B.1d depicts run 6(green) where the maximum number of steps per episode was increased from 100 to 200. This change was intended to give the agent more time to learn the sequential logic of the level—collecting all chips and then reaching the exit. However, the results were nearly identical to those in Figure B.1c, run 5(magenta), indicating that extending the episode length alone was insufficient for improving learning outcomes.

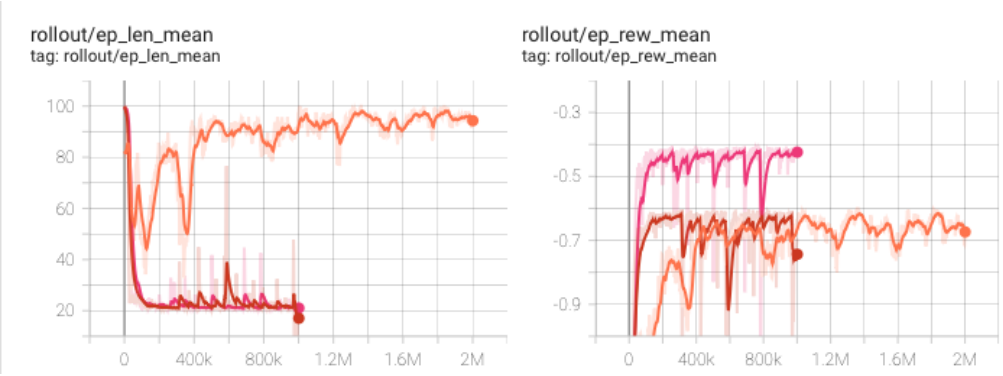
Figure B.2 shows the resulting agent's trajectories in visual way. We can observe for run 3 and run 5, agent's last position is right to the pink hazardous tile, indicating the backfire triggered in the training session.



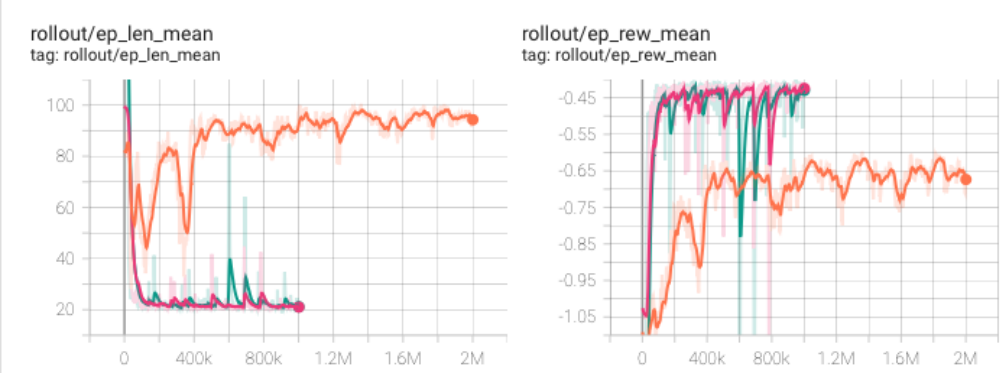
(a) Run 1 base agent



(b) Runs 2 and 4 with initial negative reward every step

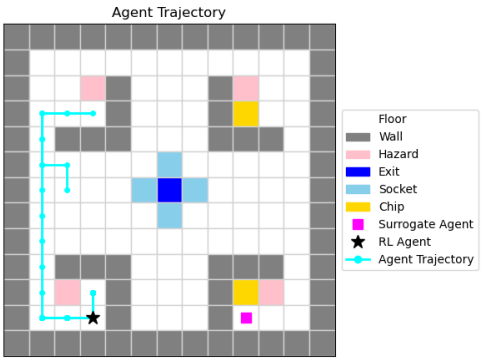


(c) Runs 3 and 5 with increased negative reward every step

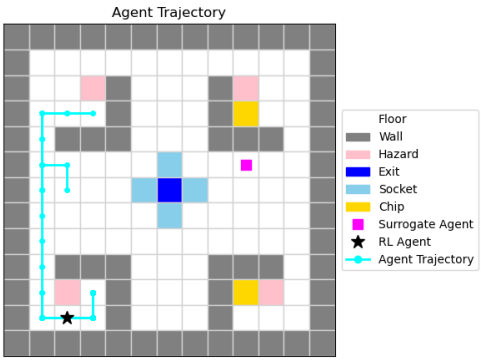


(d) Run 6 with increase maximum steps per episode

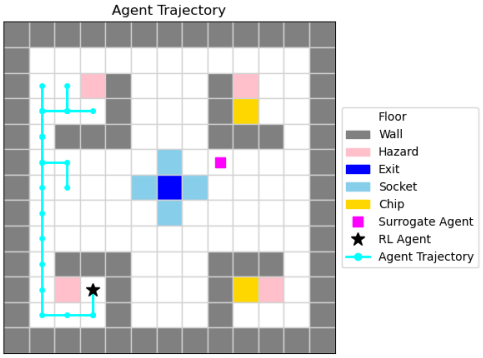
Figure B.1: Training progress across different experimental conditions.



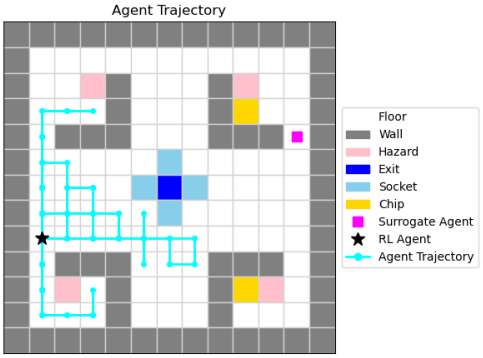
(a) Run 1 base agent trajectory



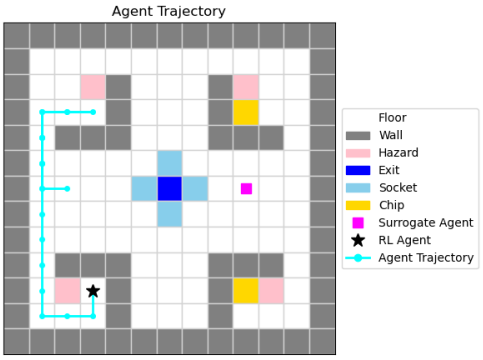
(b) Run 2 agent trajectory



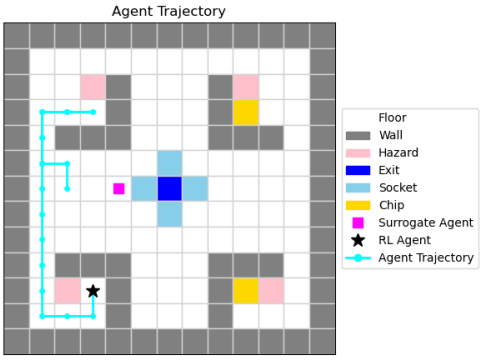
(c) Run 3 agent trajectory



(d) Run 4 agent trajectory



(e) Run 5 agent trajectory



(f) Run 6 agent trajectory

Figure B.2: Evaluation trajectories of trained agents across six different runs