REPORT

Petros Georgoulas Wraight

September 2025

1 Introduction

1.1 Observation Space

We represent the $M \times N$ grid as a **three-channel binary tensor** so $o_t \in \{0,1\}^{3\times M\times N}$. The three channels are spatial masks that encode the full task state needed for navigation while keeping the representation compact and consistent.

Layer semantics.

- **Channel 1 (agent):** a single 1 at the agent's current cell; 0 elsewhere.
- **Channel 2 (goal):** a single 1 at the goal cell; 0 elsewhere.
- **Channel 3 (obstacles):** 1s at obstacle cells; 0 elsewhere.

Thus, $\sum o_t^{(1)} = 1$, $\sum o_t^{(2)} = 1$, and $\sum o_t^{(3)} = K$ (the number of obstacles). This yields a fully observable grid encoding without auxiliary features or normalization. This representation is simple enough for multilayer perceptron networks (chosen in this project for simplicity), yet compatible with convolutional networks if desired (e.g. bigger grid environment).

1.2 Action Space and Dynamics

The agent selects actions from a discrete set of four cardinal moves:

$$\mathcal{A} = \{\text{up}, \text{down}, \text{left}, \text{right}\} \tag{1.1}$$

An action at time t proposes a candidate next cell by adding a direction specific offset to the current grid coordinate (r_t, c_t):

$$(r_{t+1}, c_{t+1}) = (r_t, c_t) + \Delta(a_t) \tag{1.2}$$

where $\Delta(a_t) \in \{(-1,0),(1,0),(0,-1),(0,1)\}$ encodes the chosen move.

State transition outcomes. Given the candidate cell, the environment applies the following rules:

- **Out of bounds**: If a candidate move lies outside the $M \times N$ grid, the episode terminates with a failure signal (negative reward).
- **Collision**: If the candidate coincides with an obstacle cell, the episode terminates with a failure signal (negative reward).
- **Goal reached**: If the candidate equals the goal cell, the episode terminates with a success signal (positive reward).
- **Regular move**: Otherwise, the agent moves to the candidate cell and receives a small per-step penalty, which implicitly encourages shorter paths.

Transitions within an episode are deterministic given (r_t, c_t) and a_t. Stochasticity arises from episode randomization (fresh start position, obstacle positions and goal position) which promotes generalization rather than memorization of a single grid layout.

1.3 Reward Design: Base vs Combined

Two different reward signals are used. The *base* reward encourages shortest path navigation with simple terminal signals. The *combined* reward adds a potential based shaping term that provided dense progress feedback toward the goal while respecting the task's termination rules.

Base reward. Let s_t denote the grid state (agent's position at time t), a_t a cardinal move and s_{t+1} the next state. Define constants $C_{\text{step}} > 0$, which denotes the per-step cost, $C_{\text{fail}} > 0$, which is the failure penalty and $R_{\text{goal}} > 0$ denote the goal reward. The base reward r_b is

$$r_{\text{base}}(s_t, a_t, s_{t+1}) = \begin{cases} R_{\text{goal}}, & \text{if } s_{t+1} \text{ is the goal state,} \\ -C_{\text{fail}}, & \text{if } s_{t+1} \text{ is out of bounds or collides with an obstacle,} \\ -C_{\text{step}}, & \text{otherwise (regular move).} \end{cases} \tag{1.3}$$

Shaping term (progress potential). Let $d(s)$ be the obstacle aware shortest path distance from the agent's cell in state s to the goal. In practise d is calculated by a breadth-first search that respects cells which contain an obstacle. Define a shaping weight $\alpha > 0$ and a shaping discount $\gamma_s \in (0, 1]$. The shaping reward for a transition in then

$$r_s(s_t, a_t, s_{t+1}) = \mathbf{1}_{\text{valid}}(s_t, a_t, s_{t+1}) a\big(d(s_t) - \gamma_s d(s_{t+1})\big) \tag{1.4}$$

where $\mathbf{1}_{\text{valid}}$ is 1 only for non-failure transitions. This yields a positive signal when a move reduces geodesic distance to the goal and avoids shaping on invalid terminations.

Combined reward. The *combined* reward simply adds the shaping term to the base reward:

$$r_c(s_t, a_t, s_{t+1}) = r_b(s_t, a_t, s_{t+1}) + r_s(s_t, a_t, s_{t+1}) \tag{1.5}$$

Intuitively, the base component enforces task correctness (reach the goal and avoid invalid moves), while the shaping component supplies dense, directionally informative feedback that should accelerate learning on randomized layouts.

The scale $a > 0$ sets the influence of shaping relative to the base signal: larger a should speed learning but risks teaching the agent to take risky paths (e.g. trying to pass between two obstacles), while a very small a makes shaping negligible. The shaping discount $\gamma_s \in (0, 1]$ controls how strongly the next state is subtracted: values near 1 strongly reward progress towards the goal and strongly penalize steps that increase the geodesic distance. A smaller γ_s makes the signal more forgiving of occasional detours from the geodesic optimal path.

The motivation for adding the shaping signal is that the base signal is sparse (most steps yield only a uniform, small cost and informative feedback appears mainly at termination). This means that on larger grids and/or cluttered layouts, exploration becomes difficult. The progress term injects dense guidance at every valid step.

Despite the aforementioned benefits, shaping in practise seems that should be applied with caution: Because the r_s term depends only on the geodesic distance, it can bias the agent towards choosing risky narrow gaps or skim obstacle edges.

Finally, because the layout (start, goal, obstacles) is fixed within an episode, the geodesic distance map $d(\cdot)$ is computed only once per episode.

1.4 The PPO and DQN algorithms

What DQN and PPO are. An on-policy method (PPO) and one off-policy method (DQN) are used. On-policy means the learner updates from rollouts generated by the *current* policy and then discards that data after some optimization steps. Off-policy means the learner can update from a replay buffer containing past experience (data collected using an old policy).

Network inputs and outputs. Observations are 3-layer binary grids, as discussed in Sec. 1.1. For training the observations are flattened to a single feature vector, so the input dimensionality is $3MN$. Under the default $M = N = 6$, that is $3 \times 6 \times 6 = 108$ features. The discrete action space has four cardinal moves (see Sec 1.2), so *action related* network outputs have size 4. In more detail PPO's policy head outputs a categorical distribution over the 4 actions and its value head outputs a single state value scalar. DQN outputs a vector with length 4 of Q values (i.e. $Q(s, a) \in \mathbb{R}^4$.

Some details about PPO. PPO collects short rollouts from vectorized environments and performs several epochs of minibatch updates on that fixed batch, then gathers fresh data. It uses a shared feature extractor with two heads: a *policy* head that parameterized the action distribution and a *value* head that estimates the state value used for advantage computation. Clipping in the objective constrains how far the new policy can move from the behavior policy per update, improving stability. In this setup, PPO trains with multiple parallel environments and a flattened observation interface suitable for an MLP policy.

Some details about DQN. DQN trains a single neural network that maps the flattened observation to $Q(s, a) \in \mathbb{R}^4$ selects actions via ϵ-greedy exploration. Transitions are stored in a replay buffer and sampled uniformly for SGD updates; a separate, periodically synchronized *target network* provides stable bootstrapping targets. The training loop interleaves environment steps with bursts of gradient updates, with an exploration schedule that decays

2 Results (Base Environment)

Table 1: Base-environment evaluation: mean episode length (steps) and success rate (%). Metrics are averaged over 50 random layouts (100 episodes per layout) to counter variance.

Algorithm	Reward	Avg. length	Success rate
PPO	Base	6.69	87.98%
PPO	Combined (shaped)	6.37	89.26%
DQN	Base	19.93	69.06%
DQN	Combined (shaped)	11.16	82.46%

Table 1 reports mean performance over 50 randomly sampled configurations, where each agent performed 100 episodes in each configuration. For each algorithm we evaluate the two reward regimes: the *base* reward and the *combined* reward. When comparing base vs. combined for a given algorithm, hyperparameters are held fixed, so differences reflect the reward only.

The observed gap in Table 1 likely reflects a combination of stability and sensitivity. PPO's clipped updates and advantage estimation make each batch of fresh, on-policy data highly usable. DQN, by contrast, bootstraps from its own evolving Q estimates and relies on replay to decorrelate experience. In a setting where the map resets every episode, the replay buffer is dominated by short, low reward transitions early in training. Finally, as encountered during this project, DQN's performance was way more sensitive to hyperparameters than PPO's. Insufficient tuning (e.g. replay buffer size, warm-up steps, exploration decay...) plausibly impacts the results seen.

The observed PPO lead is robust to reasonable setting changes, but stronger DQN tuning would probably narrow the gap. I therefore interpret these results as evidence of *practical* superiority under the chosen budget and settings, and not a universal dominance claim.

3 Expanded Environment

Observation space. The base observation is extended to a four channel binary tensor $o_t \in \{0, 1\}^{4\times N\times M}$. The first three layers mirror the base setup (see Section 1.1), while the *fourth layer marks the* remaining *bonus positions* required before the goal can pay out. Once a bonus cell is first visited, it is removed from this layer, so the observation space always reflects the remaining sub-goals.

Action space and dynamics The action space remains the four cardinal moves. Termination conditions are identical to the base environment (e.g. goal collision, out of bounds), with one refinement: *reaching the goal without having collected all bonuses ends the episode with a penalty.* This makes the task explicitly hierarchical in the sense that the agent should first collect all bonuses and then he can reach the final goal destination.

Reward function Let s_t be the agent position at time t, g the goal cell, $\mathcal{B} = \{b_1, \ldots, b_m\}$ the set of bonus cells, and $Z_t \subseteq \mathcal{B}$ the set of bonuses *already* visited (first visit only). With step penalty $c_{\text{step}} > 0$, failure penalty $C_{\text{fail}} > 0$,

first visit bonus R_{bonus}, goal reward R_{goal} and premature goal penalty P_{prem}, the per step reward is given by:

$$r_t = -c_{\text{step}} + R_{\text{bonus}} \sum_{i=1}^{m} \mathbf{1}\{s_{t+1} = b_i \wedge b_i \notin Z_t\} + R_{\text{goal}} \mathbf{1}\{s_{t+1} = g \wedge |Z_{t+1}| = m\} \tag{3.1}$$

$$- P_{\text{prem}} \mathbf{1}\{s_{t+1} = g \wedge |Z_{t+1}| < m\} - C_{\text{fail}} \mathbf{1}\{\text{collision or out of bounds at } t+1\} \tag{3.2}$$

Training and evaluation metrics for this task the PPO algorithm is selected. The evaluation inference tracks standard metrics and additionally the *average number of bonuses visited* to quantify subgoal completion.

4 An Alternative Approach: Absorbing-MDP formulation

Motivation. We can cast the grid world as a finite Markov decision process (MDP) with explicit *absorbing* states for goal, obstacles and out of bounds (OOD) positions. In this formulation, once the agent enters any absorbing state, he remains there forever and has zero reward thereafter. This formulation lets us apply dynamic programming methods (e.g. value iteration) on a fixed layout.

State, action and absorbing sets. Let the grid be $\mathcal{G} = \{1, \dots, M\} \times \{1, \dots, N\}$ with obstacles $\mathcal{O} \subset \mathcal{G}$, free cells $\mathcal{F} = \mathcal{G} \setminus \mathcal{O}$ and goal cell $g \in \mathcal{F}$. Define the *OOB* cells (the cells reachable from $\mathcal{G}$ by one cardinal move) as

$$\partial\mathcal{G} = \{(0, c), (M+1, c) : 1 \le c \le N\} \cup \{(r, 0), (r, N+1) : 1 \le r \le M\}$$

Now the extended state space is

$$\tilde{S} = \mathcal{F} \cup \mathcal{O} \cup \partial\mathcal{G}$$

with the absorbing set

$$Abs = \{g\} \cup \mathcal{O} \cup \partial\mathcal{G}$$

Transition function $p(s', r|s, a)$. the probability of next state/reward pairs is defined as follows:

1. **Absorbing states:** for any $\tilde{s} \in Abs$ and any action $a \in \mathcal{A}$, $p(\tilde{s}, 0|\tilde{s}, a) = 1$. Therefore, once at goal, an obstacle, or an OOB cell, the agent stays there with zero reward forever.
2. **Non-absorbing states:** For any $s \in \mathcal{F} \setminus \{g\}$ and $a \in \mathcal{A}$, let $y = \nu(s, a)$ be the (deterministic) next position. With step penalty c_{step}, failure penalty C_{fail}, and goal reward $R_{\text{goal}} > 0$,

$$p(s', r \mid s, a) = \begin{cases} 1 & \text{if } s' = y \in \mathcal{F} \setminus \{g\}, \quad r = -c_{\text{step}}, \\ 1 & \text{if } s' = g, \quad r = R_{\text{goal}} - c_{\text{step}}, \\ 1 & \text{if } s' \in \mathcal{O} \cup \partial\mathcal{G} \text{ and } s' = y, \quad r = -c_{\text{step}} - C_{\text{fail}}, \\ 0 & \text{otherwise.} \end{cases}$$

   regular moves incur the per-step cost; stepping into the goal yields the goal bonus (and then the process becomes absorbing at g); attempting to enter an obstacle or cross the boundary moves the agent into the corresponding absorbing failure state with a one-time failure penalty.

Using the above formulation, value iteration (VI) computes:

$$V_{k+1}(s) = \begin{cases} 0, & s \in \mathcal{A}\lfloor\int, \\ \max\limits_{a \in \mathcal{A}} \sum\limits_{s', r} p(s', r \mid s, a) \left[r + \gamma V_k(s') \right], & s \in \mathcal{F} \setminus \{g\}. \end{cases}$$

For a given layout, VI iteration converges to some optimal value function $V^\star(s)$. Given this function, then the agent selects the greedy one-step lookahead with respect to $V^\star$

$$\pi^\star(s) \in \arg\max_a \sum_{s', r} p(s', r|s, a)[r + \gamma V^\star(s)]$$

Table 2: Value Iteration on Randomized Grid Layouts. For each of 100 randomly generated layouts, value iteration was solved to obtain V^*, then an agent starting from a random position followed the greedy policy induced by V^* for evaluation.

Metric	**Value**
Number of layouts	100
Success rate	99%
Average return	6.57
Average trajectory length	4.23

Table 2 shows that value iteration yields near-perfect performance across randomized layouts: the agent only fails when the goal is truly unreachable (i.e., obstacles and boundaries partition the grid so there is no legal path from the start region to the goal). Still, one might choose model-free RL (PPO/DQN) even when $p(s', r|s, a)$ is known. a learned policy helps us plan over a *distribution* of layouts (we don't resolve the problem, during test time), scales to large or continuous observation spaces where exact DP is impractical and tolerates modeling noise. In short, planning is optimal for a fixed map; model-free training spends compute upfront to produce a reusable policy for many maps.