RATATOUILLE: Reward-driven Adaptive Training And Task-oriented Optimization Utilizing Intrinsic Learning Labyrinth Exploration

Hung Le
University of Chicago
Chicago, IL 60637
conghunglt@uchicago.edu

Andre de la Cruz
University of Chicago
Chicago, IL 60637
andredlcruz@uchicago.edu

Daniel Chen
University of Chicago
Chicago, IL 60637
dchn@uchicago.edu

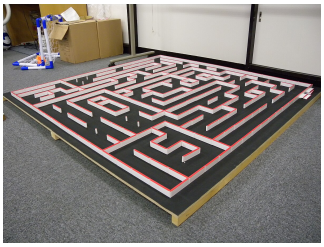
Abstract

RATATOUILLE is a reinforcement learning-driven formulation of the Micromouse task. We propose a distance-based exponential reward signal that guides the agent towards the goal as quickly as possible while penalizing collisions. Our training pipeline incorporates Soft-Actor Critic (SAC), Prioritized Experience Replay (PER) and curriculum learning to enhance efficiency. We evaluate our approach on a variety of fixed, hand-crafted mazes. The code and environment are publicly available at `https://github.com/andred1729/Ratatouille`.

1 Introduction

Reinforcement learning has achieved great success in continuous-control domains, but it has notably struggled in maze-like settings due to sparse reward environments and inefficient exploration. To explore this problem, we draw inspiration from the famous Micromouse robotics competition, in which teams attempt to build an autonomous "mouse" that can traverse from the bottom-left corner to the center of an unseen maze as quickly as possible, and the shortest run to reach the goal wins.

Figure 1: A standard 16×16 Micromouse maze.



Current approaches to Micromouse look for the quickest path, preferring long straights where the mouse can build velocities to shorter paths that have many turns. However, Micromouse paths still follow strict straight lines, and do not use the width of the track to its full capacity. We expect that a machine learning approach will be able to take advantage of this.

To find the most effective way to solve a maze, we introduce RATATOUILLE, a framework that couples the Soft Actor–Critic algorithm [1] with reward shaping based on an exponential distance curve and an incremental training curriculum, bound together with a replay memory using Prioritized Experience Replay (PER) [2]. Specifically, we define a bounded exponential function over the normalized distance-to-goal to provide smooth gradients even when far from the center, and we

Final Project for TTIC 31170: Robot Learning and Estimation (Spring 2025).

heavily penalize wall collisions. We further dynamically adjust the maximum episode length as well as the effective maze size, enabling the agent to master shorter episodes and simpler layouts before tackling more complex mazes.

Our paper includes:

- A distance-based reward shaping mechanism that is an effective densification of the sparse Micromouse reward.
- Integration of PER into the SAC framework for an efficient maze-solving agent.
- Ablation studies on how various factors affect the training process.
- An adaptive curriculum on episode length and effective maze size to guide learning from easy to hard tasks.

2 Environment

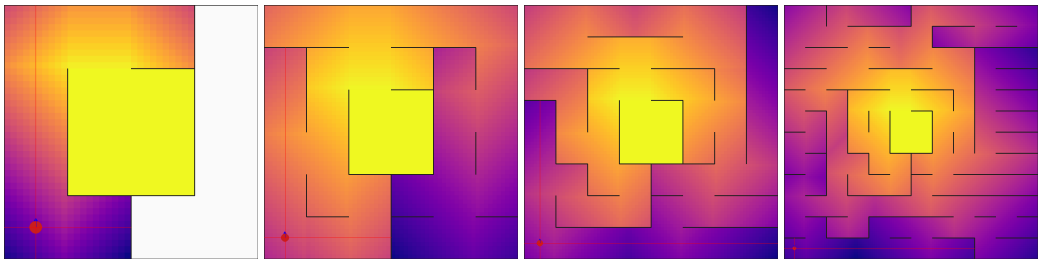
2.1 **Standard Micromouse vs** RATATOUILLE

In this paper, we use robot, mouse and rat interchangeably. We first preface with the important differences between standard Micromouse and our formulation of it. In the standard Micromouse task, the maze is unseen, and the rat has to perform maze exploration to figure out the optimal path. In RATATOUILLE the maze layout is a priori known and fixed to us (the human programmers), but we do not include any inductive bias in our algorithm and simply craft universal reward signals to guide and encourage exploration. Another distinction is that in the standard Micromouse task, the true state (position, velocity, etc.) are not accessible to the rat — most rats do some kind of estimation via fusing data from motor encoders, IR sensors, IMUs and so on. In our formulation, the environment gives the rat its true state.

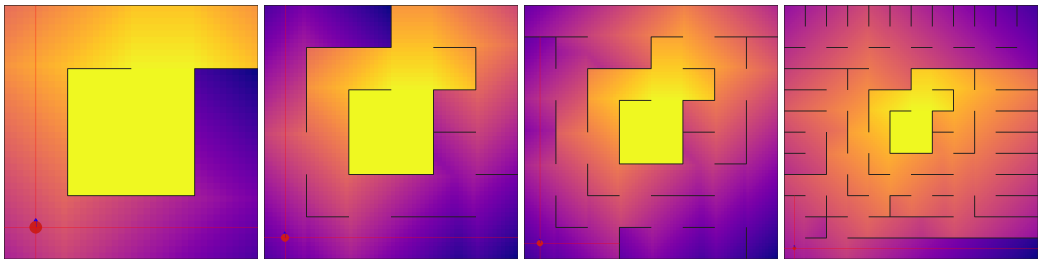
2.2 RATATOUILLE **environments**

To this end, we have hand crafted 8 mazes as follows.

Figure 2: RATATOUILLE mazes.



(a) 4×4-1 (b) 6x6-1 (c) 8x8-1 (d) 12x12-1



(e) 4×4-2 (f) 6x6-2 (g) 8x8-2 (h) 12x12-2

The mazes are designed in two series to aid in cirriculum training. In both series, each previous maze is nested in the center of the next maze. For example, the 4x4-1 maze is copied in the center of the 6x6-1 maze, which is then copied in the center of the 8x8-1 maze, and so on.

In all mazes, the rat starts at the bottom-left corner, while the goal region is fixed to the central 2×2 area, which is highlighted in yellow. The surrounding grid shows a gradient that visualizes the shortest-path distance to the goal, with cells closer to the goal appearing brighter and those farther away shaded progressively darker purple.

2.3 The Markov Decision Process

Let n be the size of the maze, and c_{LiDAR} be the number of LiDAR beams the robot has access to.

We use state and observation interchangeably because the process is fully observable. We adopt the standard MDP formulation with the tuple $\langle \mathcal{S}, \mathcal{A}, \gamma, P, R \rangle$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, γ is the discount factor, $P(s'|s,a)$ is the state transition function, and $R(s,a,s')$ is the reward. We made the following design choices so that our environment mimics true Micromouse conditions.

The reference frame we adopt is such that the center of the maze has coordinates $(0,0)$, and each square in the maze corresponds to 1 unit, and the agent is a circle of radius 0.1. Each state is represented as a vector including the robot's Cartesian coordinates, orientation in radians, left and right wheel velocities, and c_{LiDAR} distance readings measuring proximity to the nearest wall in c_{LiDAR}-evenly spaced out directions. Actions correspond to continuous accelerations applied to each wheel, bounded within the interval $[-1,1]$. Our states and actions are therefore represented as

$$s = \left[x,\, y,\, \theta,\, v_L,\, v_R,\, \text{lidar}(x,y,\theta)^\top\right]^\top \in \mathbb{R}^{5+c_{LiDAR}} =: \mathcal{S}, \quad a = [a_L,\, a_R]^\top \in \mathbb{R}^2 =: \mathcal{A},$$

where $\text{lidar}(x,y,\theta) = [\ell_1, \ldots, \ell_{c_{LiDAR}}]^\top \in \mathbb{R}^{c_{LiDAR}}$ are the LiDAR distance readings, i.e., the distance from the robot to the nearest wall in the absolute headings $\left\{\theta + \frac{2\pi k}{c_{LiDAR}} : 0 \le k \le c_{LiDAR} - 1\right\}$. The robot is deterministically initialized at

$$s_0 = \left[\frac{-(n-1)}{2}, \frac{-(n-1)}{2}, \frac{\pi}{2}, 0, 0, \text{lidar}\left(\frac{-(n-1)}{2}, \frac{-(n-1)}{2}, \frac{\pi}{2}\right)^\top\right]^\top,$$

which is at the center of the bottom left cell of the maze, facing north and stationary.

2.4 State transition

As the agent interacts with the environment through acceleration on the left and right motors, the environment updates the position and orientation of the robot using its current wheel velocities v_L and v_R, following the kinematics of differential wheeled robots. Noise is not added, i.e., there is no slippage modeled, and the robot's inputs are applied exactly with a cap on the maximum velocity. LiDAR distances are also noiselessly calculated based on the robot's position and orientation. As such, the motion of the robot can be described through standard odometry calculations with transition

$$P(s'|s,a) = \delta_{\overline{s}},$$

a point mass at $\overline{s}$ where $\overline{s} = \left[\overline{x},\, \overline{y},\, \overline{\theta},\, \overline{v_L},\, \overline{v_R},\, \text{lidar}(\overline{x},\overline{y},\overline{\theta})^\top\right]^\top$ with

$$\begin{aligned}
\overline{v_L} &= \text{bound}(v_L + a_L \Delta t, v_{\max}), & \overline{v_R} &= \text{bound}(v_R + a_R \Delta t, v_{\max}),\\
\overline{x} &= x + \cos\theta \left(\frac{v_L + v_R}{2}\right)\Delta t, & \overline{y} &= x + \sin\theta \left(\frac{v_L + v_R}{2}\right)\Delta t,\\
\overline{\theta} &= \left(\frac{v_R - v_L}{b}\right)\Delta t,
\end{aligned}$$

where

- $\text{bound}(v,m) := \max\{\min\{v,m\}, -m\}$ bounds the magnitude of the first argument within the second argument. In our case, $v_{\max} = 0.4$.
- $\Delta t = 0.1$ is the duration of a step in the simulation.
- and $b = 0.2$ is the distance between the "wheels" of the robot.

The exception to above is when the simulation terminates: either the maximum episode length has been reached, the robot has collided with the wall, or the robot has reached the center of the maze.

2.5 Reward shaping

The standard approach to the Micromouse task is to measure the floodfill distance $d_1(s)$ from state s to the maze center, which is really just the length of the shortest path from the robot's current cell to the center cells, when viewing the maze with cells as vertices as an undirected graph in the natural sense. The obvious thing to do is to make our reward signals based off of this distance — however this signal is still quite sparse, e.g., the robot is unable to tell apart similar locations in the same cell. To alleviate this and densify the reward signal, we further divide up each cell into 10×10 sub-cells, and again compute the floodfill distance $d_{10}(s)$ in this new graph. The subscripts in d_1 and d_{10} are therefore natural, as we can view the original maze as a trivial 1×1 division of each cell. We then define our non-terminal rewards to be

$$r(s, a, s') = \frac{e^{k(1-\frac{d_1}{\max d_1})} - e^k}{e^k - 1} + \frac{e^{k(1-\frac{d_{10}}{\max d_{10}})} - e^k}{e^k - 1}$$

where we take inspiration from that $x \mapsto \frac{e^{kx}-e^k}{e^k-1}$ increases from -1 at $x = 0$ to 0 at $x = 1$ in an exponential-like manner. The parameter k controls the steepness of the curve. We chose $k = 2$. It is worth noting that these non-terminal rewards are bounded in $[-2, 0]$, in particular nonpositive, hence discouraging lingering around certain cells. Previous iterations of our reward shaping included giving positive reward when some sense of progress is made — it was then empirically observed that sometimes the agent would prefer to linger and stay stationary around these states and not actually attempt to complete the maze — hence perhaps more fine-tuned reward signals that also take into account stationarity were called for, but we opted for this option of setting generally negative rewards instead.

For terminal rewards, since we desire to make the reward shaping size-agnostic, we define

$$r(\text{wall collision}, \cdot, \cdot) = nr_{wall}, \quad r(\text{center}, \cdot, \cdot) = nr_{center},$$

where we recall that n is the size of the maze we're concerned about, and r_{wall} is a (large) negative constant, and r_{center} is a (large) positive constant. It is helpful for now to think of $r_{wall} = -25, r_{center} = 25$ in a 4×4 maze.

3 Reinforcement learning methodology

3.1 Soft Actor-Critic (SAC)

Our method builds on the standard SAC implementation with automated temperature tuning. At a high level, the critic comprises two parallel networks that estimate Q-values for state–action pairs, while the actor outputs distributional parameters for a diagonal Gaussian policy, from which actions are sampled and then projected onto the environment's valid action space. In particular, we have an actor network π_θ, two critic networks Q_{ϕ_1}, Q_{ϕ_2}, two target critic networks $Q_{\phi_{targ,1}}, Q_{\phi_{targ,2}}$ that are initialized as Q_{ϕ_1} and Q_{ϕ_2} respectively, and a tunable temperature parameter α.

Then at each training step, a batch $\{(s_i, a_i, s'_i, r_i, d_i) : i \in [B]\}$ of B transitions is sampled according to PER (see below) and the critic networks are updated with pessimistic target Q estimates

$$a'_i \sim \pi_\theta(\cdot, s'_i), \quad y_i = r_i + \gamma(1 - d_i)\Big[\min_{j\in\{1,2\}} Q_{\phi_{targ,j}}(s'_i, a'_i) - \alpha \log \pi_\theta(a'_i \mid s'_i)\Big],$$

to minimize the Bellman loss

$$\mathcal{L}_{\text{critic}}(\phi_j) = \frac{1}{B}\sum_{i=1}^{B} w_i\big(Q_{\phi_j}(s_i, a_i) - y_i\big)^2 \quad \text{for } j = 1, 2.$$

where w_i are importance sampling weights to be determined by PER.

On the other hand, we update the actor by minimizing

$$\mathcal{L}_{\text{actor}}(\theta) = \frac{1}{B}\sum_{i=1}^{B}\big[\alpha \log \pi_\theta(a'_i \mid s_i) - \min_{j\in\{1,2\}} Q_{\phi_j}(s_i, a_i)\big],$$

and the temperature parameter by minimizing

$$\mathcal{L}_{temp}(\alpha) = \frac{1}{B}\sum_{i=1}^{B}\Big[\alpha\big(-\log \pi_\theta(a_i \mid s_i) - \mathcal{H}_{\text{target}}\big)\Big],$$

where $\mathcal{H}_{target}$ is the target entropy. Finally, the target critics get soft updates with

$$\phi_{targ,j} \leftarrow \tau\,\phi_j + (1-\tau)\phi_{targ,j} \quad \text{for } j = 1, 2.$$

The notable thing about the SAC architecture is that it balances exploration and exploitation through entropy regularization, in the form of the $-\alpha \log \pi_\theta$ term. The temperature parameter α is also automaticallly tuned as seen above to keep the actor's entropy not too low (too deterministic) and not too high (too random), and simply around some target entropy $\mathcal{H}_{target}$.

3.2 Prioritized Experience Replay (PER)

As alluded to above, in our training we used PER to sample from our replay buffer, instead of uniformly sampling. New online transitions are automatically stored in the replay buffer with the then-maximum priority value in the buffer. Otherwise, in each training step, the priority for each transition in the batch is updated as the average magnitude of the two critics' temporal-difference errors

$$p_i = \frac{|\delta_{i,1}| + |\delta_{i,2}|}{2} \tag{1}$$

where

$$\delta_{i,j} = |Q_{\phi_j(s_i,a_i)} - y_i| \quad \text{for } i \in [|\mathcal{D}|], j = 1, 2,$$

which are computed at little to no overhead as they were already computed during the forward pass and in Q updates.

At every new training step, sampling probabilities are set proportional to these scores with

$$P(i) = \frac{p_i^\alpha}{\sum_j p_j^\alpha} \quad \text{for } i \in [|\mathcal{D}|].$$

This however yields bias in our Q estimates, which must then be corrected via importance sampling weights

$$w_i = \big(N\,P(i)\big)^{-\beta} \quad \text{for } i \in [|\mathcal{D}|],$$

that are then normalized to only scale down updates, not up

$$w_i \leftarrow \frac{w_i}{\max_j w_j}.$$

3.3 Curriculum training

We developed our curriculum training through an incremental training schedule in both time and size. The first approach entails dividing the total training horizon into four equal quartiles, where in the first quartile, training episodes are capped at 50% of the maximum length, then 70% in the second quartile, 90% in the third quartile and lastly 100% in the final quartile. In this paper we call this approach **maximum episode length curriculum**. The goal of this approach is to minimize unnecessary and confusing long excursions at the beginning.

The second approach entails starting the robot (at every environment reset) closer to the goal in the earlier quartiles, and gradually move to the original, true starting location in later quartiles. For example, to train for a 12×12 maze, we would first start at the bottom-left corner of the center 4×4, then to that of the center 6×6, 8×8 and eventually 12×12. In this paper, we call this approach **initial state curriculum**. In virtue, this approach says that "you should learn to do the easy tasks first, before learning to do the harder ones". In practice, we also implemented another approach that roughly models this in virtue — which is to pretrain the model on a smaller maze (say, 6×6), then initialize the agent on a bigger maze (say, 8×8) with the weights of the smaller model to facilitate learning. This is in virtue similar to the initial state curriculum above, as our hand-crafted mazes are cross-similar, so the 6×6 does look similar to the center 6×6 of the 8×8. The sole significant

difference between these two implementations is that the first implementation has a lot less training time, namely a third or a fourth of a training run while the second implementation spends the entire training run on each "maze size". We will come to see later that this difference will come in crucial, but we believe that these two implementations are fundamentally the same. For us, due to time constraints, this is still unresolved.

3.4 Implementation details

A single global random seed ensures reproducibility across our entire codebase. Our hyperparameters are as in Table 1.

Table 1: RATATOUILLE hyperparameters.

Hyperparameter	**Value**
Number of training steps	500000
Number of evaluation episodes	10
Optimizer	Adam
Batch size	256
Discount factor (γ)	0.99
Soft-update factor (τ)	0.005
Learning rate	3×10^{-4}
Target entropy $\mathcal{H}_{target}$	$-\frac{1}{2}\dim(\mathcal{A}) = -1$
Prioritization exponents	$\alpha = 0.3$, β from 0.4 to 1

During evaluation, we keep track of 3 metrics. In decreasing order of importance, they are: average completion rate, average episode length and average episode return.

3.5 Manual control and visualization

Our codebase also includes an optional human-in-the-loop allows for manual joystick-style control for debugging, and with a Pygame module displaying the maze, robot pose, and sensor readings in real time during training and evaluation.

4 Results

In all of our results below, unless mentioned otherwise, our training uses PER and maximum episode length curriculum, with $r_{wall} = -25, r_{center} = 25$ and $c_{LiDAR} = 4$. The shaded area in all graphs denote one standard error of the mean over 3 seeds.

4.1 Small mazes

Figure 3: Completion rate on small mazes.

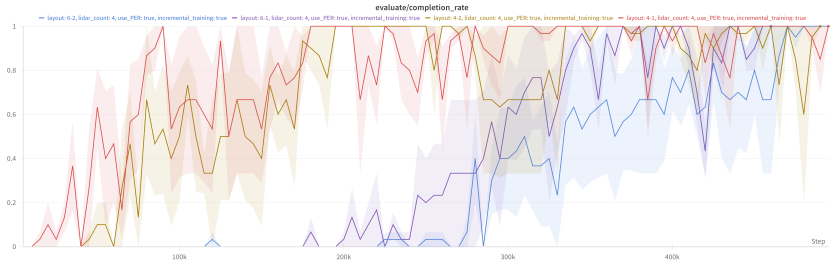


Figure 3, Figure 4 and Figure 5 indicate the completion rate, the average episode length and average episode return respectively, for runs on 4×4-1 (in red), 4×4-2 (in brown), 6×6-1 (in purple) and 6×6-2 (in blue). It is easy to see that as the maze gets progressively harder, the completion rate and

Figure 4: Average episode length on small mazes.

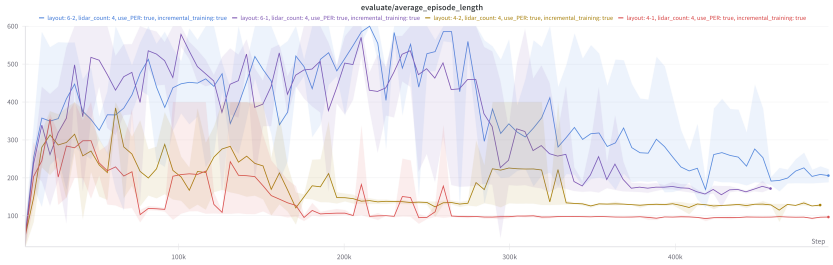
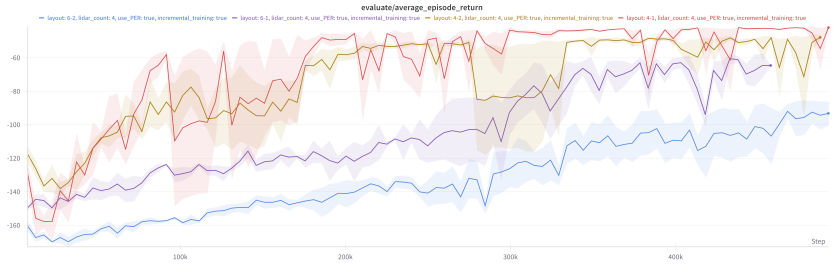


Figure 5: Average episode return on small mazes.



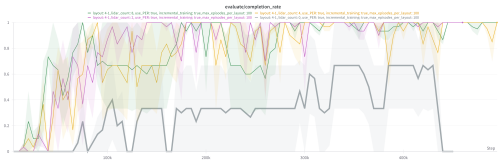
the average episode return picks up later in the training loop, with the average episode length also stabilizing later, and notably to a higher value. This conforms with our notion of difficulty — it does indeed takes longer for the agent to say, do the 6×6-1 maze than the 4×4-2 maze.

Overall, for all of these "small mazes", the agent converges to an optimal path successfully with training from scratch.

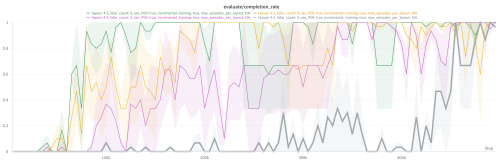
4.2 Ablation studies on number of LiDAR beams: c_{LiDAR}

With our primary goal of solving the maze achieved, we then questioned how reliant the agent is on the LiDAR readings. In particular, if it is given a different set of readings, would it still be able to learn, and if so, a lot slower or quicker? Figure 6 below provides a comparison when only c_{LiDAR} is varied among $\{0, 2, 4, 8\}$ in various small mazes, where we pay particular attention to the situation when $c_{LiDAR} = 0$ in gray.

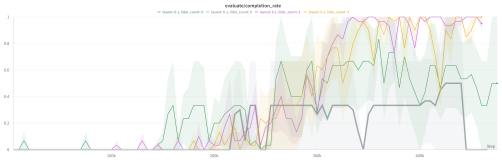
Figure 6: Completion rate with varying c_{LiDAR} on small mazes. (Gray: 0, Pink: 2, Orange: 4, Green: 8)



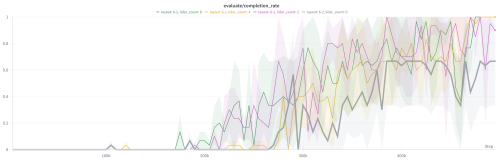
(a) 4×4-1



(b) 4×4-2



(c) 6×6-1



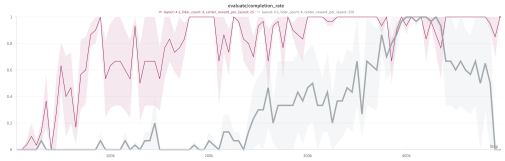
(d) 6×6-2

As observed across all small mazes, setting $c_{LiDAR} = 0$ significantly diminishes learning, not just in the completion rate sense as shown in the figure above, i.e., the agent can only consistently solve much later, but also in longer average episode length. As for $c_{LiDAR} \in \{2, 4, 8\}$, we observe that there is no statistically significant difference in their performance. We therefore stick (and have stuck) to $c_{LiDAR} = 4$ across all our experiments partially due to this, and partially due to that it best mimics the standard Micromouse setting.

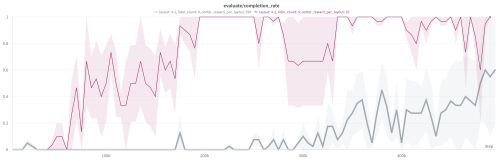
4.3 Ablation studies on terminal reward scaling: r_{wall} and r_{center}

It is intuitive to incur a large cost when the agent collides with the wall, as well as to give out a large reward when the agent center. What is unintuitive, as we find out below, is that setting these values to extreme magnitudes might in fact diminish learning, instead of aiding learning.

Figure 7: Completion rate with 2 settings of (r_{wall}, r_{center}) on small mazes. (Gray: Large magnitude, Pink: Regular magnitude)

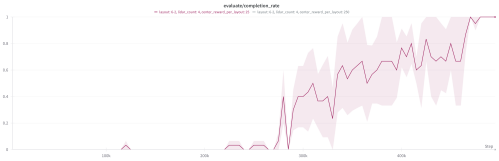


(a) 4×4-1



(b) 4×4-2

(c) 6×6-1



(d) 6×6-2

In Figure 7, the regular magnitude setting corresponds to when $r_{wall} = -25, r_{center} = 25$ as we have discussed so far. The large magnitude setting corresponds to when $r_{wall} = -250, r_{center} = 250$. As observed, overly large crash/goal rewards actually hurt learning stability and speed compared to moderate rewards. It is worth recalling that non-terminal, progress-related rewards are on the order of $[-2, 0]$. We offer a possible explanation as follows, where we inspect the critic loss of the runs on 4×4-1. In Figure 8 we can observe that critic loss here effectively does not converge, in particular,

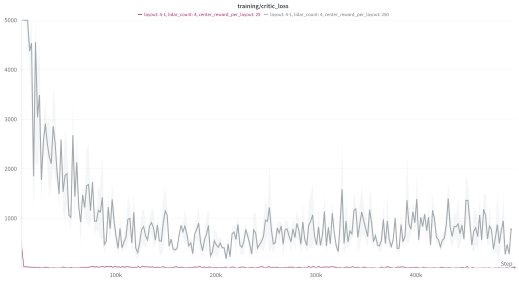


Figure 8: Critic loss for large magnitude on 4×4-1.

since we keep the same learning rate. It seems that having drastically different wall/center rewards (as compared to non-terminal rewards) would require the Q networks to approximate drastically different orders of magnitude for slightly different states, i.e., highly discontinuous. Obviously, this is only a heuristic.

4.4 Ablation studies on PER and maximum episode length curriculum

Surprisingly, removing either PER or the maximum episode length curriculum yields only minor performance drops, showing graceful degradation to individual component removal.

4.5 Scaling up: big mazes with initial state curriculum

For bigger mazes (8×8-1, 8×8-2, 12×12-1, 12×12-2), it turns out that training from scratch for 500000 steps as-is does not work.

A non-result that we also observed is that doing initial state curriculum (over 500000 steps) also did not help. What we mean by that is for example, on the 8×8-1, training the agent 166K steps on the center 4×4, then 166K steps on the center 6×6, then finally 166K steps on the entire 8×8-1 does not help. Same for 12×12, where we step up from 4, 6, 8 and then 12.

However, solving 8×8-2 from scratch with the agent initialized with the weights from a 6×6-2 model that was pretrained for 500K steps was successful. Then, solving 12×12-2 with the aforementioned pretrained 8×8-2 model was also successful. As remarked in subsection 3.3, this is actually in virtue the same thing as doing initial state curriculum — however we have not had time to look into this further.

5 Conclusion

Our approach of coupling the SAC architecture with PER consistently achieves strong performance on all 4×4 and 6×6 maze layouts, converging in fewer than 400K steps. Additionally, our ablation studies confirm that the system is robust, showing that the agent's performance does not overly depend on any single component, given the same LiDAR counts and suitable environment reward signals.

Although our model performed well in smaller mazes, we did encounter challenges when generalizing to larger mazes. We expect that improving reward shaping and using better methods to bootstrap learning from smaller mazes will greatly enhance our results in this area. Hence our next steps include:

- **Better reward shaping**: Some inefficient behaviors from our agent were observed, which could be addressed through refined rewards.
- **Improved bootstrapping from smaller mazes**: Using offline data, rather than directly transferring weights, might lead to more stable and effective learning.
- **Generalizing to unseen mazes**: Successfully extending the agent's capabilities to new, unseen maze structures remains a particularly rewarding goal.

In conclusion, we have demonstrated that modern continuous-control reinforcement learning, combined with thoughtful reward shaping and lightweight curriculum techniques, can effectively tackle maze navigation tasks. We believe this can meaningfully contribute to the broader challenge of maze-solving in RL and might even find applications in interactive scenarios similar to RATATOUILLE.

Acknowledgments and Disclosure of Funding

This work was done as a final project for the class TTIC 31170: Robot Estimation and Learning in Spring 2025. The authors would like to thank Professor Matt Walter for a fruitful and engaging course, and for the opportunity to explore of a topic of our interest. Despite the long hours and considerable effort invested, the work was conducted without compensation.

References

[1] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018. URL `https://arxiv.org/abs/1801.01290`.

[2] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2016. URL `https://arxiv.org/abs/1511.05952`.