Olympia, Francheska L. Layos, Miguel Raphael
Alis, Renz Andrei C. Manzanero, Brix Anthony G.

reinFOURcement BSCS – ML COM22

LOCK-N-KEY

INTRODUCTION

One of the fundamental challenges of reinforcement learning is making the agent learn quickly and optimally in complex and dynamic environments through trial and error. This short study focuses on developing a simple maze-type environment game that allows the agent to complete a sequential type of goal, from retrieving the key at a certain grid, and then unlocking the door, while avoiding the enemy and different obstacles.

This setup portrays real-world decision-making scenarios where the agent must know how to plan actions strategically and quickly, prioritizing the objectives, and adapting to the environment's uncertainty or complexity. Addressing these challenges provides a better understanding of the reinforcement learning algorithms chosen to test throughout the project.

Lock-and-Key serves as a straightforward practical and good testing setting on where strategies and decision-making are crucial. Every action performed by the agent has consequences, much like in real life. It may result in failure or move it closer to the objective. This game demonstrates how reinforcement learning allows an agent to gain knowledge from previous actions, develop over time and determine the most effective strategy for success.

METHODOLOGY

This study evaluates the performance of established reinforcement learning (RL) control methods, specifically Q-Learning, Monte Carlo (MC), and Actor-Critic (AC), within a custom-designed grid-world environment that features increasing complexity and a hybrid reward structure. The methodology utilizes discrete-state and discrete-action formulations, enabling direct comparison between tabular and semi-tabular learning approaches.

Environment

The environment, designated as Lock-N-Key, is a custom implementation utilizing the Gymnasium framework and simulates a 6 by 6 grid navigation puzzle. This configuration enables detailed analysis of agent behavior as environmental difficulty increases incrementally.

State and Action Space

- Environment Grid: The environment consists of a constrained 6 by 6 grid that incorporates a fixed arrangement of wall obstacles, thereby restricting available pathways for agent navigation.

- Observation Space: The state variable s is formally defined as a discrete space, Discrete (6^6). The agent receives an observation array that includes the coordinates of key elements and a binary indicator variable.

$$s = \left[r_A, C_A, r_K, C_K, r_L, c_L, has_{key}\right]$$

Here, r and c represent the row and column coordinates for the Agent (A), Key (K), and Lock (L). In the agent implementation, the continuous observation vector is converted into a hashable tuple to facilitate indexing of tabular Q-functions or value functions.

- Action Space: The agent operates in a Discrete (4) action space, defined as$A = \{0, 1, 2, 3\}$. Actions 0 through 3 correspond to deterministic movements (right, left, up, down).

Hybrid Reward Function

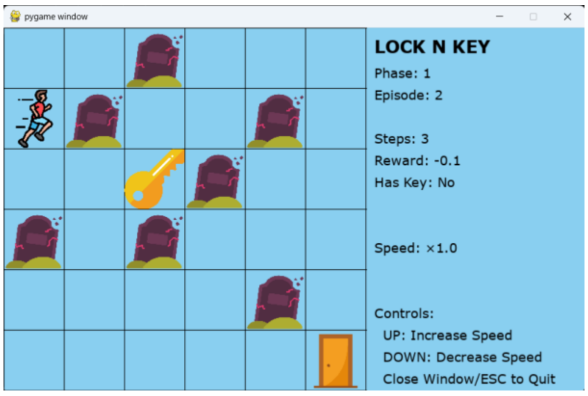
The reward function $R(s, a, s')$ combines sparse, goal-oriented rewards with continuous reward shaping to facilitate faster learning, especially during advanced phases of increased difficulty.

Event	**Value**	**Description**
STEP_PENALTY	-0.1	Standard penalty to encourage path efficiency.
KEY_REWARD	+50.0	Stepping onto the Key tile (occurs once).
DOOR_REWARD	+100.0	Stepping onto the Door tile with Key (occurs once).
CAUGHT_PENALTY	-25.0	Collision with the Enemy (Phase 5 only, episode termination)
MAX_STEPS	200	Episodes are truncated in 200 steps to manage training time.
SURVIVAL_BONUS	0.00	Survival of the agent, neutral setting. No gains nor losses for staying alive
DIST_SCALE	0.1	How far is the agent from the enemy to encourage the agent to increase its distance from the enemy
APPROACH_SCALE	0.1	Reward system for moving closer to the objectives (key and door) allows to learn directional improvement

Environment Phases

The experiment systematically varies the environment's complexity across five phases to test the agents' generalization capabilities:

Phase	**Walls**	**Agent Start**	**Key Position**	**Lock Position (Door)**	**Enemy (Hazard)**
1	Fixed	Fixed ([0, 0])	Fixed ([2, 2])	Fixed ([5, 5])	Inactive
2	Fixed	Random Free Cell	Random Free Cell	Random Free Cell	Inactive
3	Random	Random Free Cell	Random Free Cell	Random Free Cell	Active (Follows trail of agent 2-3 steps behind)



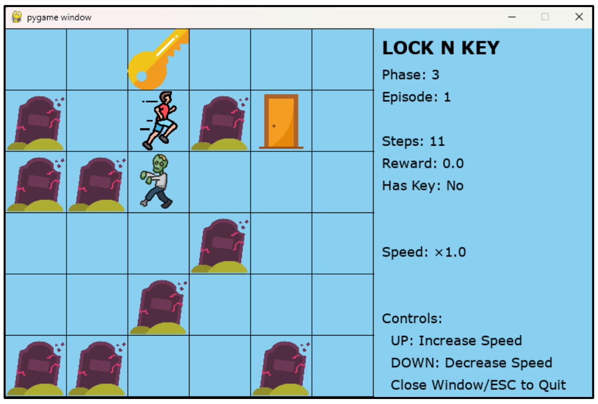
Phase: 1
Episode: 2

Steps: 3
Reward: -0.1
Has Key: No

Speed: ×1.0

Controls:
UP: Increase Speed
DOWN: Decrease Speed
Close Window/ESC to Quit

Figure 1: Snippet Training UI Phase 1



Phase: 3
Episode: 1

Steps: 11
Reward: 0.0
Has Key: No

Speed: ×1.0

Controls:
UP: Increase Speed
DOWN: Decrease Speed
Close Window/ESC to Quit

Figure 2: Snippet Training UI Phase 3

Reinforcement Learning Algorithms

A. Tabular Q-Learning Agent – a value-based approach that learns the best action for each state through trial and error, using rewards as the environment

B. Incremental Monte Carlo (MC) Agent – A learning algorithm that estimates action values from the average of rewards from completed episodes, updating its values after each episode completed

C. Tabular Actor-Critic (AC) Agent – A hybrid agent that combines the policy learning of actor and value estimation of critic to update both after every step for a faster and more stable learning

Training and Analysis

The training process is managed by the *train_and_visualize* function, which standardizes both execution and data collection procedures.

A. Execution and Persistence

1. User Interface: Training parameters are selected through an interactive Pygame menu, which facilitates the tuning of hyperparameters.

2. Visualization: A Pygame window displays the grid-world simulation in real-time, featuring a menu
   - Pick phase (1-3)
   - Choose algo
   - Number of episodes
   - Hyper parameter tuning
     - Q learning
       - Alpha
       - Gamma
       - Epsilon
     - Monter Carlo
       - MC learning rate
       - Gamma

        - Epsilon
    - Actor-Critic
        - Alpha (Actor LR)
        - Beta (Critic LR)
        - Gamma

3. Policy Storage: Upon completion of training, the final learned policy, whether a Q-table or V/π tables, is saved using the Python pickle library to the file lockkey_policy.pkl for future evaluation or deployment.

Performance Evaluation

1. Learning Curves: Performance is evaluated visually using matplotlib plots, which display the training trajectory across the total number of episodes.
    - Raw Episode Rewards: Plots the total reward achieved in each episode.
    - Smoothed Episode Rewards: A moving average, typically with a window size of N = 10, is applied to the raw rewards to highlight learning trends and reduce high-variance fluctuations.
    - Cumulative Success Rate: This metric measures the proportion of successful episodes relative to the total number of episodes completed.
2. Metric Reporting: The final summary presents the highest performance values achieved for the following key metrics.
    - Best Raw Reward
    - Best Smoothed Reward
    - Best Cumulative Success Rate
    - Total Episode Lengths

RESULTS

The game created consists of 3 phases wherein each phase was tested with the three chosen algorithms and different parameters. Each algorithm and phase are tested with the given parameters:

	Learning Rate (α)	**Discount Factor (γ)**	**Exploration Rate (ε)**
Default Parameters	0.10	0.90	0.20
High Learning Rate (α)	0.90	0.90	0.20
Low Learning Rate (α)	0.01	0.90	0.20
High Exploration Rate (ε)	0.10	0.90	0.90
Low Exploration Rate (ε)	0.10	0.90	0.05
High Discount Factor (γ)	0.10	0.99	0.20
Low Discount Factor (γ)	0.10	0.50	0.20

While in the Actor-Critic Algorithm, the agent does not just choose the best action, it goes through the policy wherein the exploration is already given into the policy itself. Below are the chosen parameters tested to evaluate the efficiency of the algorithm:

	Actor Learning Rate (α)	**Critic Learning Rate (β)**	**Discount Factor (γ)**
Default Parameters	0.10	0.10	0.90
High Actor LR (α)	0.30	0.03	0.90
Low Actor LR (α)	0.03	0.03	0.90
High Discount Factor (γ)	0.10	0.01	0.99
Low Discount Factor (γ)	0.10	0.01	0.50
High Critic LR (β)	0.10	0.50	0.90
Low Critic LR (β)	0.10	0.01	0.90

Table 1: Q-Learning results per parameter and phase

Q-LEARNING PHASE 1				
	Best Raw Reward	**Best Smooth Reward**	**Best Success Rate**	**Average Episode Length (Steps)**
Default	150.30	150.30	1.00	10.96
High LR (α)	150.30	150.30	1.00	10.94
Low LR (α)	150.30	150.30	1.00	11.02
High ER (ε)	150.30	150.26	1.00	20.09
Low ER (ε)	150.30	150.30	1.00	10.29
High DF (γ)	150.30	150.30	1.00	10.92
Low DF (y)	150.30	150.30	1.00	10.94

Q-LEARNING PHASE 2				
	Best Raw Reward	**Best Smooth Reward**	**Best Success Rate**	**Average Episode Length (Steps)**
Default	150.21	147.81	1.00	60.35
High LR (α)	150.22	148.20	1.00	57.35
Low LR (α)	150.20	148.01	1.00	58.56
High ER (ε)	150.40	148.20	1.00	58.40
Low ER (ε)	150.20	147.82	1.00	61.05
High DF (γ)	150.30	148.17	1.00	61.05
Low DF (y)	150.12	148.11	1.00	61.70

Q-LEARNING PHASE 3				
	Best Raw Reward	Best Smooth Reward	Best Success Rate	Average Episode Length (Steps)
Default	150.24	145.09	1.00	82.52
High LR (α)	150.20	147.09	1.00	80.66
Low LR (α)	150.04	144.08	1.00	89.23
High ER (ε)	149.62	141.76	1.00	136.52
Low ER (ε)	150.20	147.44	1.00	77.56
High DF (γ)	149.74	146.17	1.00	85.61
Low DF (y)	150.10	146.90	0.82	89.02

In the results of phases 1 – 3 with different parameter settings, it can be seen that a Low Exploration Rate (ε) is better from all with demonstrating a low average step which means the agent finished the goal faster than other parameter setting, thinking of it for application and testing, it is not recommended to use a low exploration rate due to it's little or lack of exploration itself around the environment. With this knowledge and consideration, the best parameter setting for Q-Learning here is a High Exploration Rate (ε) for it also shows a good metric evaluation and better compatibility in terms of testing phase of algorithms.

Table 2: Monte Carlo results per parameter and phase

MONTE CARLO PHASE 1				
	Best Raw Reward	Best Smooth Reward	Best Success Rate	Average Episode Length (Steps)
Default	150.30	150.10	0.97	22.15
High LR (α)	150.30	150.30	0.97	16.06
Low LR (α)	150.30	150.30	0.98	14.58
High ER (ε)	150.30	150.27	1.00	20.20
Low ER (ε)	150.30	150.30	0.90	28.90
High DF (γ)	150.30	150.30	0.96	18.73
Low DF (y)	150.30	150.30	0.91	29.14

MONTE CARLO PHASE 2				
	Best Raw Reward	Best Smooth Reward	Best Success Rate	Average Episode Length (Steps)
Default	150.10	50.06	0.13	191.41
High LR (α)	150.21	48.27	0.20	197.48
Low LR (α)	150.12	48.10	0.11	192.42
High ER (ε)	149.54	116.10	0.50	163.51
Low ER (ε)	148.28	12.30	0.04	198.88
High DF (γ)	149.50	65.47	0.08	192.54
Low DF (y)	148.80	62.86	0.06	194.70

MONTE CARLO PHASE 3				
	Best Raw Reward	Best Smooth Reward	Best Success Rate	Average Episode Length (Steps)
Default	149.01	22.36	0.50	181.41
High LR (α)	149.41	29.07	0.13	182.54
Low LR (α)	149.84	28.66	0.13	183.40
High ER (ε)	149.94	113.93	0.47	150.65
Low ER (ε)	149.90	19.83	0.02	185.73
High DF (γ)	149.80	45.39	0.17	181.25
Low DF (y)	150.10	30.92	0.11	183.78

The Monte Carlo Algorithm shows a good evaluation in phase 1, but as it comes to encountering harder or a more complex game environment, the algorithm itself starts to struggle with the training time taking longest among the three. The metric and learning curve describing that for this game, this algorithm ranks the lowest for its output.

Table 3: Actor-Critic results per parameter and phase

ACTOR-CRITIC PHASE 1				
	Best Raw Reward	Best Smooth Reward	Best Success Rate	Average Episode Length (Steps)
Default	149.79	149.79	1.00	14.84
High LR (α)	149.79	149.79	1.00	14.71
Low LR (α)	149.79	149.79	0.98	21.38
High DF (γ)	149.79	149.79	0.98	19.72
Low DF (y)	149.79	149.79	0.97	32.37
High LR (β)	149.59	149.59	1.00	15.22
Low LR (β)	149.99	149.88	1.00	12.62

ACTOR-CRITIC PHASE 2				
	Best Raw Reward	Best Smooth Reward	Best Success Rate	Average Episode Length (Steps)
Default	149.52	140.22	1.00	154.88
High LR (α)	150.02	118.27	0.53	169.48
Low LR (α)	149.74	118.35	0.75	161.43
High DF (γ)	149.51	117.20	0.73	162.80
Low DF (y)	149.10	116.22	0.63	168.88
High LR (β)	150.30	130.47	0.56	153.54
Low LR (β)	149.12	130.47	0.54	156.17

ACTOR-CRITIC PHASE 3				
	Best Raw Reward	Best Smooth Reward	Best Success Rate	Average Episode Length (Steps)
Default	150.17	125.54	0.69	147.83
High LR (α)	149.765	125.53	1.00	148.21
Low LR (α)	149.82	123.89	1.00	153.01
High DF (γ)	150.10	113.01	0.42	155.40
Low DF (y)	150.12	127.13	1.00	159.68
High LR (β)	149.96	140.30	0.70	123.04
Low LR (β)	149.62	118.68	0.54	148.79

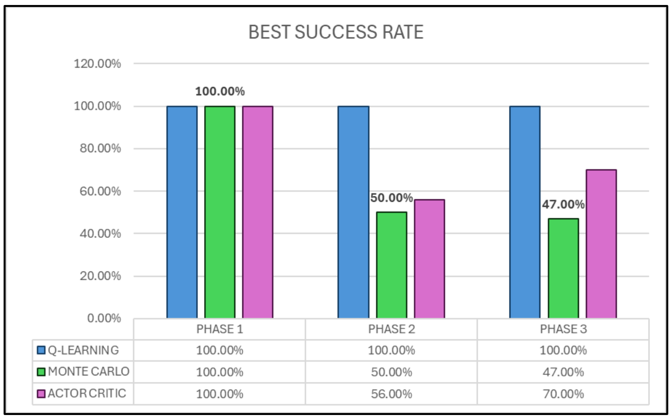
Lastly, when it comes to Actor-Critic algorithm, phase 1 shows a good performance for mostly all parameter settings, it showed promising policy and value functionality with a fixed environment (Phase 1). At phase 3, the agent struggled with many low success rates, looking at the best parameter setting for phase 5, it would be the High Critic Learning Rate (β) where the agent is showing a better stability with a better success rate stability according to the evaluation graph.

DISCUSSION

With training each algorithm chosen with different parameters set to all the 3 phases of the game to know make the game more competitive for the learning, it can be seen from the chart that overall, from the steps to the success rate, the Q-Learning Algorithm performs well to our Lock-N-Key game. The success rate is used for identifying on how the agent achieves its defined goal, from collecting the key to unlocking the door or completing the level out of all the attempts made with the formula:

$$Success\ Rate = \frac{Number\ of\ Successful\ Episodes}{Total\ Number\ of\ Episodes}$$

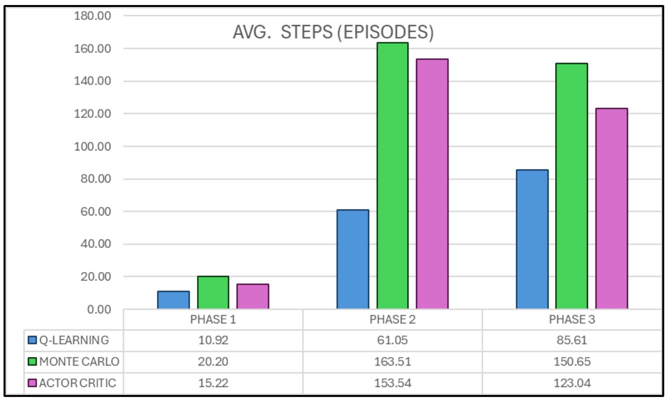
Monte Carlo struggled deeply as the game became more complex or difficult, with just achieving a 47% best success rate in phase 3. This discusses that in this game environment and setup; Monte Carlo cannot accomplish the target goal. On the other hand, Q-Learning shows a very promising success rate at which at a certain point in the episode training, the goal was achieved by the agent. Aside from the success rate, the average length of episodes (by steps) was also measured with the formula:



	PHASE 1	PHASE 2	PHASE 3
🟦 Q-LEARNING	100.00%	100.00%	100.00%
🟩 MONTE CARLO	100.00%	50.00%	47.00%
🟪 ACTOR CRITIC	100.00%	56.00%	70.00%

Figure 3: Best Success Rate

$$Avg\ Length\ of\ Episodes = \frac{Total\ Steps\ Taken\ in\ All\ Episodes}{Number\ of\ Episodes}$$



	PHASE 1	PHASE 2	PHASE 3
🟦 Q-LEARNING	10.92	61.05	85.61
🟩 MONTE CARLO	20.20	163.51	150.65
🟪 ACTOR CRITIC	15.22	153.54	123.04

Figure 4: Avg. Steps

This evaluation metric is used to understand how long the agent wander around the grid or environment does while completing the said objective, completed or not. The shorter the average length, the faster wherein it means that the policy is more efficient, while the longer indicates that the agent explores too much which may lead to struggling or inefficient paths. In Q-Learning, the algorithm is more efficient and is faster in completing the goal. Overall, Q-Learning performs better followed by Actor-Critic and lastly, Monte Carlo.

Q-Learning shows the best evaluation results among the three which is why it is the main algorithm used in the game environment Lock-N-Key. The best parameter used in this is the High Discount Factor (γ) Parameter Setting where the Learning Rate (α) = 0.1, Discount Factor (γ) = 0.9 and Exploration Rate (ε) = 0.90. In this parameter setting, it can be seen in the chart that efficiency and the consistency of the algorithm, from the smooth reward to raw reward, the success rate over the whole training episode, and the episode lengths.

Q-Learning High Discount Factor (γ) Learning Rate (α) = 0.10 ; Discount Factor (γ) = 0.99 ; Exploration Rate (ε) = 0.20				
	Best Raw Reward	Best Smooth Reward	Best Success Rate	Avg. Episode Length (Steps)
Phase 1	150.30	150.30	1.00	10.92
Phase 2	150.30	148.17	1.00	61.05
Phase 3	149.74	146.17	1.00	85.61

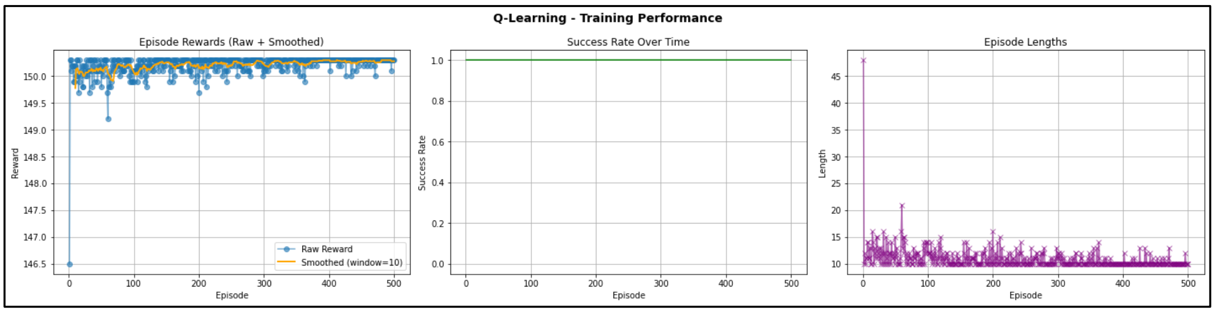


Figure 5: Q-Learning High Discount Factor Parameters Phase 1 Training Performance

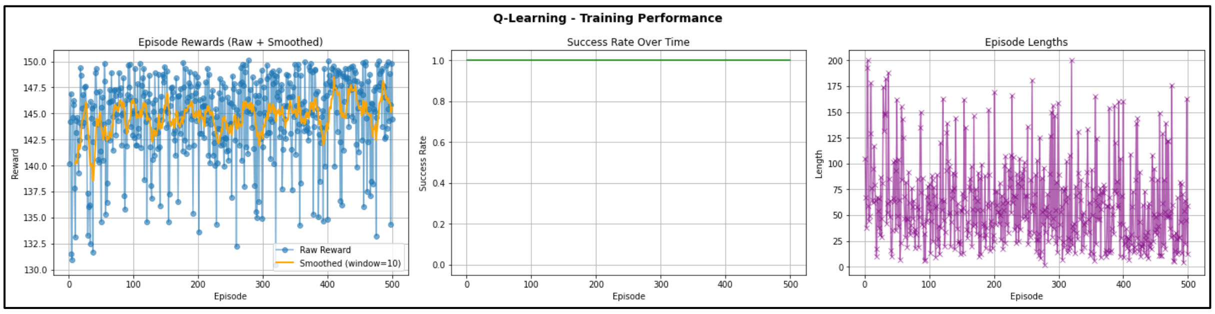


Figure 6: Q-Learning High Discount Factor Parameters Phase 2 Training Performance

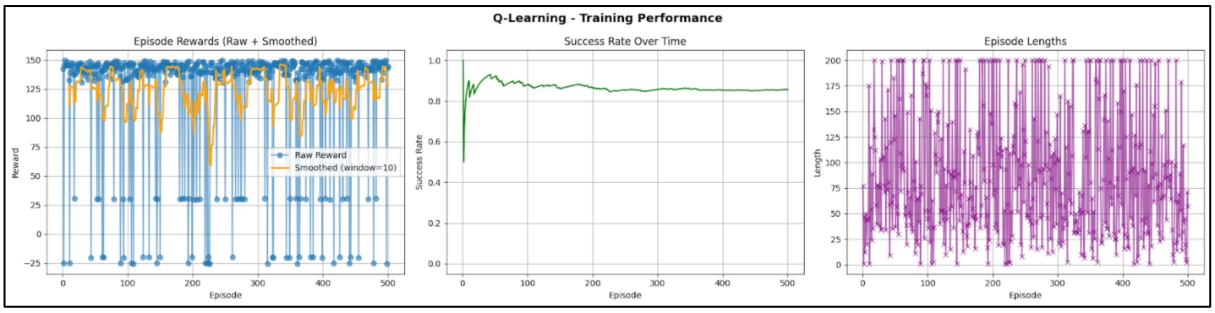


Figure 7: Q-Learning High Discount Factor Parameters Phase 3 Training Performance

Overall, Q-Learning achieved the highest success and efficiency from all training episodes and phases, with a high discount factor serving as a reminder that the agent values the future rewards more than the immediate ones, it encouraged the agent and learning process of it to develop strategies and goal-oriented behavior to complete the episodes. The minimal fluctuations of the smooth and raw rewards indicate that the agent was able to learn an optimal policy and maintain it even though the environment became more complex at the next levels.

CONCLUSION

The result of this game experiment concludes that Q-Learning consistently achieved all the most stable and efficient performance across all the phases of the game environment setting. Even during the testing phase of the algorithms, Q-Learning proved its evaluation better than the two algorithms. Based on the analysis of the success rate and average episode length, Q-Learning showed a higher success rate and a faster task completion. Supported by the raw and smooth reward, Q-Learning maintained a stable performance. It highlighted its consistency while improving over time. Overall, Q-Learning ranks the highest, its value-based approach allowed the agent to learn effectively even in environments that showed delay rewards. Q-Learning with the parameter of high discount factors proves that balancing exploration and exploitation is good when it comes to maze-like game environments.

The other two algorithms could still be improved, most especially the Actor-Critic, it shows promising learning but starts to struggle as the phase or level becomes harder, meaning the more complex the game became, the higher the struggle the Actor-Critic faced. This algorithm may require longer episode training time to see the efficiency of the algorithm. On the other hand, Monte Carlo may need to be explored upon different parameters and more episode training to see a better result in the game.

Overall, the game could be enhanced more by introducing larger maps, additional obstacles for more complexity over time. The future work may focus on hyperparameter optimization to see a wider and deeper understanding of each algorithm's dynamics and adaptability.