Solve Maze with Reinforcement Learning

Achia Rosin

based on a project RL course, IDC, 2023

1 Introduction

Reinforcement learning (RL) is a type of machine learning where an agent learns to make decisions in an environment by performing certain actions and receiving rewards or penalties. we'll see how to apply it to solve a maze game. The first step is to define the environment. Here, the environment is the maze itself, including the layout of the walls and obstacles, the location of the starting point and the end goal, and any rewards or penalties that may be present. Next, we define the agent and its actions. The agent is the entity that navigates through the maze, and its actions are the moves it can make: up, down, left, or right. Once the environment and agent are defined, we can begin the training process. The agent starts in the starting position and chooses an action based on its current state and the rewards it has received in the past. As the agent moves through the maze, it receives rewards for reaching certain locations or collecting items, and penalties for hitting obstacles or going in the wrong direction. Through this process, the agent learns to make better decisions based on the rewards and penalties it receives. Over time, it will learn the most efficient path to reach the end of the maze. Finally, once the agent has been trained, it can be tested by placing it in the maze and seeing how well it can navigate to the end goal. With the right training and enough data, the agent should be able to solve the maze with high accuracy.
We make use of 4 algorithms: Policy Iteration (Dynamic Programming), Monte-Carlo, Q-Learning (Temporal Difference Method) and SARSA (Temporal Difference Method). Policy Iteration is a model based method therefore we applied it on 5x5 maze board size.Monte-Carlo, Q-Learning and SARSA applied on 15x15 and 25x25 stochastic maze boards.
A deterministic maze is where the outcome an action is certain and predictable. A stochastic maze, on the other hand, is a maze where the outcome of an action is uncertain, if the agent takes a certain action in a certain state, it may lead to different states depending on random factors. Solving a deterministic maze is generally easier than solving a stochastic maze because the agent can rely on its past experience to make decisions. However, stochastic mazes are more realistic and can better represent real-world problems, where the outcome of actions is often uncertain.

1.1 Related Works

RL book
RL github

2 Solution

2.1 General approach

Here we used 4 different RL algorithms, detailed as follow:

Policy Iteration Method:
The policy iteration (PI) method is a commonly used algorithm in RL to solve maze games. The basic idea behind policy iteration is to improve the agent's policy in each iteration, which is the decision-making process that determines the agent's actions in a given state. The PI algorithm consists of two steps: policy evaluation and policy improvement. Policy evaluation: In this step, the agent's current policy is evaluated by simulating the agent's behavior in the environment and calculating the expected return, which is the sum of the rewards the agent is expected to receive in the future.

$$V(s) = \sum_{s'} P^{\pi(s)}_{s,s'}(R_{s,s'} + \gamma V(s')) \tag{1}$$

Policy improvement: In this step, the agent's current policy is improved by finding the action that would yield the highest expected return in each state. This new policy is then used in the next iteration of the algorithm.

$$V(s) = arg\, max \sum_{s'} P^{a}_{s,s'}(R_{s,s'} + \gamma V(s')) \tag{2}$$

The algorithm repeats these two steps until the policy converges to an optimal policy, which is the policy that maximizes the expected return for the agent. One key advantage of the PI method is that it starts with an arbitrary initial policy and improves it over time, which means that it does not require a good initial policy to begin with. Additionally, it guarantees that the policy will converge to the optimal policy given enough iterations. In the context of solving a maze game, the states are the different positions the agent can be in, the actions are the possible moves the agent can make, and the rewards are the points the agent receives for reaching certain locations or collecting items. PI is a model-based algorithm, it requires a complete model of the environment, including the transition probabilities and rewards. In practice, it can be difficult to obtain such a model, and it may not scale well to large or complex environments.

Monte-Carlo Method:
The Monte Carlo (MC) method is a type of RL algorithm that can be used to solve maze games. Unlike PI method, which uses a model of the environment

to estimate the expected return, the MC method uses actual experience to estimate the return. The basic idea behind the MC method is to use random sampling to simulate the agent's behavior in the environment. The agent starts in the starting position and makes a series of random moves. After each move, the agent receives a reward or penalty based on the location it reaches. Once the agent reaches the end of the maze or a terminal state, the return is calculated as the sum of the rewards received during the episode. This return is then used to update the agent's policy. Equation of Return is given below (where $\gamma\ is\ discount\ factor$).

$$G_t = R_t + \gamma^1 R_{t+1} + \ldots + \gamma^{T-t+1} R_T \tag{3}$$

After that we use that return values to update the value of each state-action pair

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(G_t - Q(S_t, A_t)) \tag{4}$$

The Monte Carlo method has a few key advantages over other RL algorithms. For example, it does not require a model of the environment, which means that it can be used in situations where the transition probabilities and rewards are unknown or difficult to model. Additionally, it can handle non-deterministic environments, where the outcome of each action is uncertain.

Q-Learning Method:
Q-Learning (QL) is a model-free, off-policy RL algorithm that can be used to solve maze games. The basic idea behind QL is to learn the value of each action in a given state, represented by a Q-value. The Q-value for a given state-action pair is the expected cumulative reward the agent will receive if it chooses that action in that state and follows the optimal policy thereafter. QL algorithm uses a Q-table to store the Q-values for all state-action pairs. The Q-table is initially filled with random values and is updated as the agent interacts with the environment. QL algorithm consists of several steps: The agent selects an action based on the current state and its current Q-values. The agent takes the selected action and reaches the next state, and receives a reward. The Q-value for the current state-action pair is updated using the following formula:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)) \tag{5}$$

where is the learning rate, R is the reward received, is the discount factor, St+1 is the next state and a is the action with the highest Q-value in the next state. The process is repeated until the agent reaches the end of the maze or a terminal state. The Q-learning algorithm uses a greedy policy, where the agent selects the action with the highest Q-value in each state. As the agent interacts with the environment, the Q-values converge to the optimal Q-values, which correspond to the optimal policy.

SARSA Method:
The SARSA (State-Action-Reward-State-Action) method is a type of RL algorithm that can be used to solve maze games. Like Q-learning, SARSA is a model-free, off-policy algorithm that uses a Q-table to store the Q-values for all state-action pairs. However, there is a key difference in the way SARSA updates the Q-values. The SARSA algorithm consists of several steps: The agent selects an action based on the current state and its current Q-values. The agent takes the selected action and reaches the next state, and receives a reward. The agent selects the next action based on the next state and its current Q-values. The Q-value for the current state-action pair is updated using the following formula:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)) \tag{6}$$

where alpha is the learning rate, R is the reward received, is the discount factor, St+1 is the next state and At+1 is the next action. The process is repeated until the agent reaches the end of the maze or a terminal state. The key difference between SARSA and QL is that SARSA uses the next action that will be taken, rather than the action that is known to be optimal. This is known as on-policy learning, as the agent is learning from the policy it is currently following.

2.2 Design

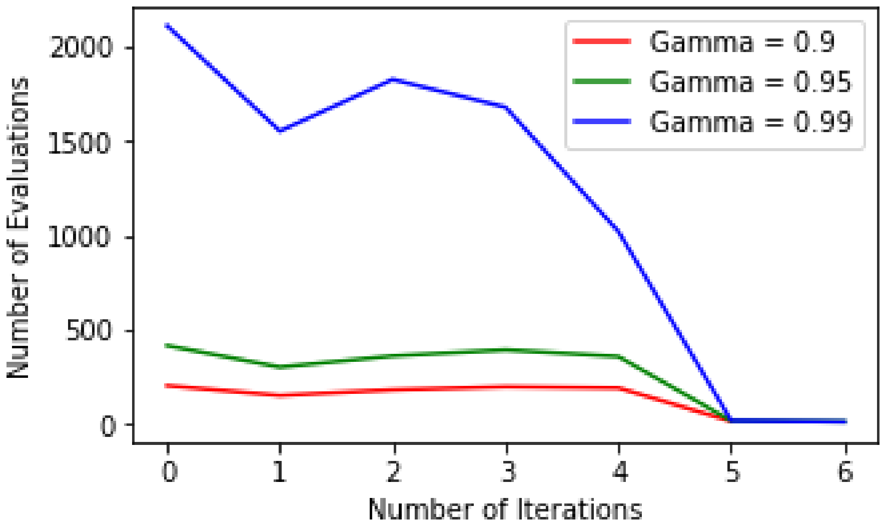
First we created a stochastic environment with probability of 0.9 of actual action is the chosen action, and 0.033 for the 3 others. accordingly we re-defined the step function. In addition we initialized the transition model, the reward matrix, and some other methods and members to use in our algorithms.

1. **Policy Iteration Method**:
   - **Policy Evaluation**: Update the values of each state according to current policy
   - **Policy Improvement**: update the policy according to max values
   - **Train**: run evaluations till stop condition, then improvement. the train stops at max episodes or when the policy no longer changes.
2. **Monte-Carlo**, **Q-Learning** and **SARSA Methods** :
   - **epsilon greedy policy**: Update the policy according to epsilon greedy
   - **update Q**: update the state-action array according to total reward of an episode
   - **Train**: let the agent run and collect data and update Q

3 Experimental results

3.1 Exercise 1

Since we don't use 'step' function because the agent is not wondering around, we'll examine the convergence by looking at the decreasing amount of evaluations per iteration. We'll plot a graph of the convergence for 3 different Discount factors:

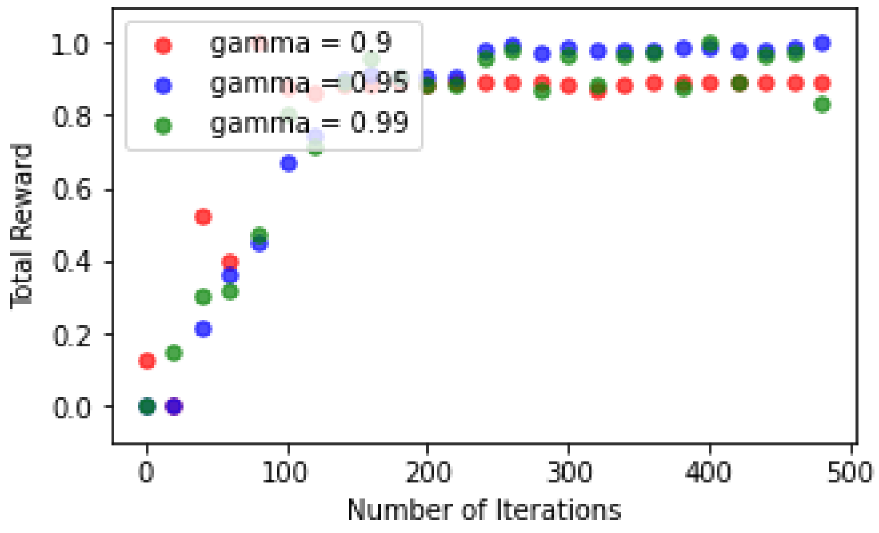


we can see that the fastest convergence is for discount factor = 0.9.

3.2 Exercise 2

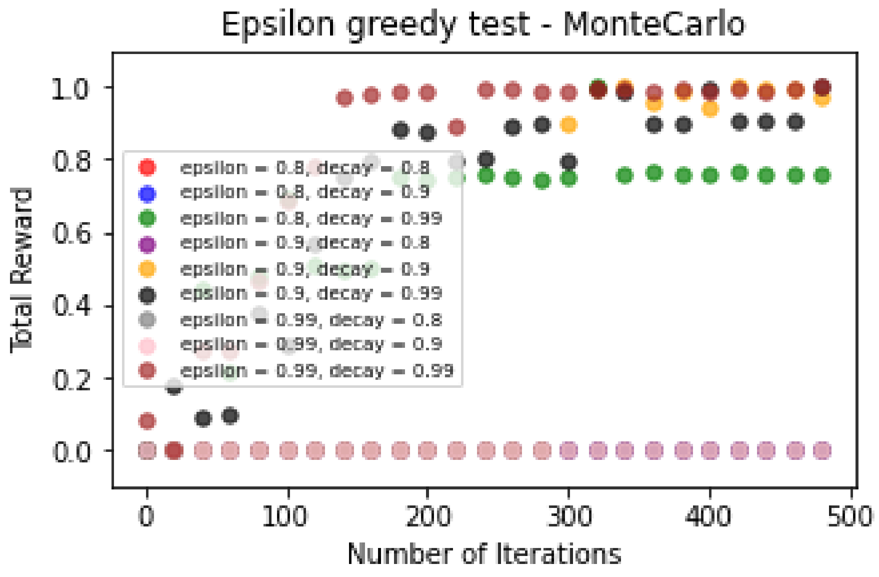
1. **Monte Carlo**:
   Explore the best parameters to get the fastest convergence under the conditions of max 500 episodes and max 500 steps, we'll test each parameter separately and continue with the best to test the next one.
   We saw that MC better get random start point for better convergence, so we let it a random start point, but at 1 of 20 it starts from (0,0). Since we measure convergence by the increasing reward, because the less the agent wandering around the bigger the reward, we can't measure it for all the 500 episodes, but for each one that starts from start point. so those are the runs we plot.
   In addition, since the algorithm is not deterministic, we'll run it a few times and get the averages of reward for each episode, so we'll have a better estimation of what's working.

   - **Discount factor**: In the first experiment we tested 3 values of discount factor upon average of 25 runs:



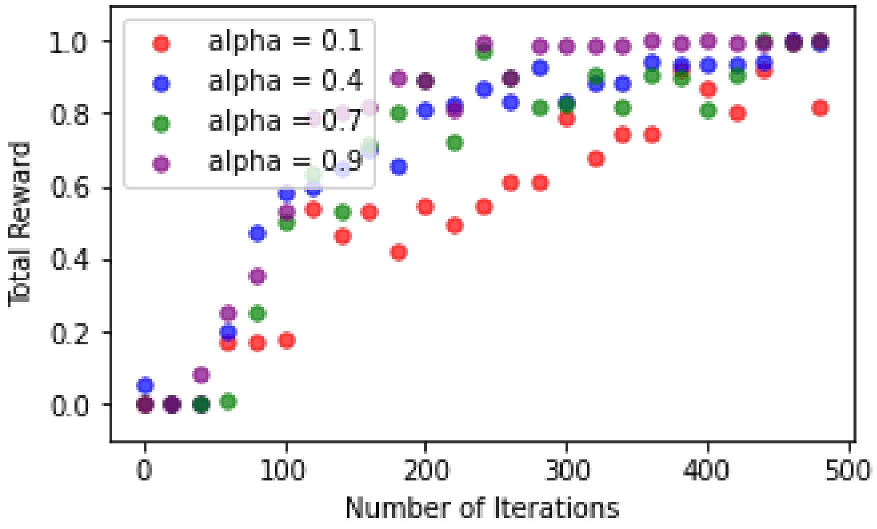
we get that we best use 0.95.

- **epsilon greedy**: we tested 3 epsilon values with 3 decay rates for each upon average of 25 runs:



we'll continue with epsilon = 0.9 and decay of (1-0.99).

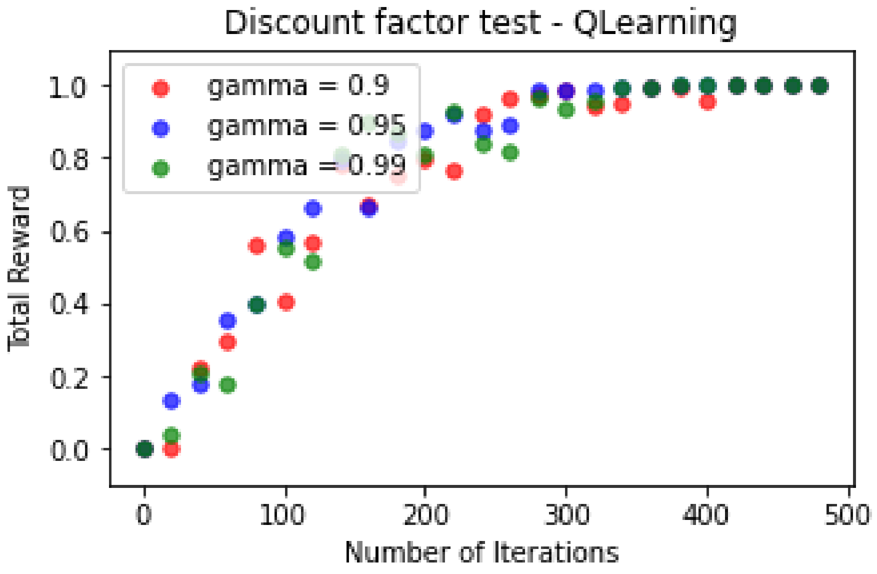
- **Alpha**: we tested 4 values upon average of 25 runs:



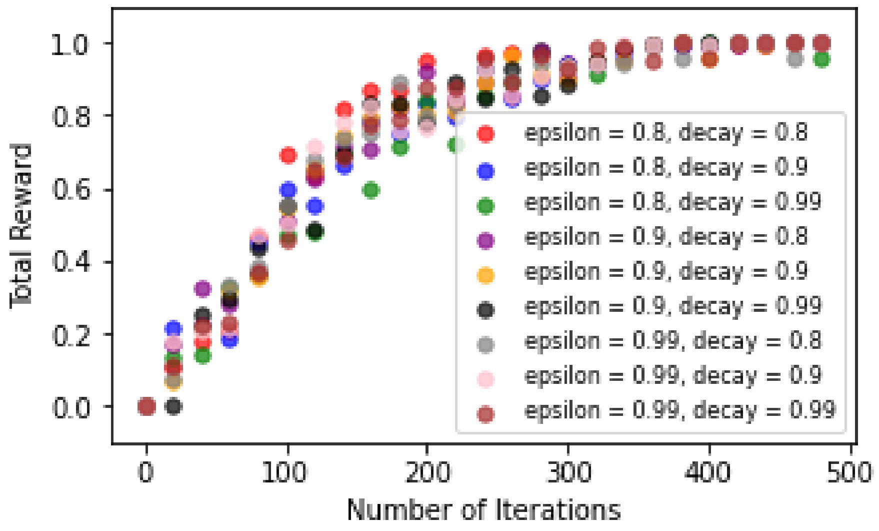
we'll continue with alpha = 0.9 and decay of (1-0.99).

2. **Q-Learning** and **SARSA** :
   For both algorithms we saw that this search of best parameters gives almost identical results, so we present them together:

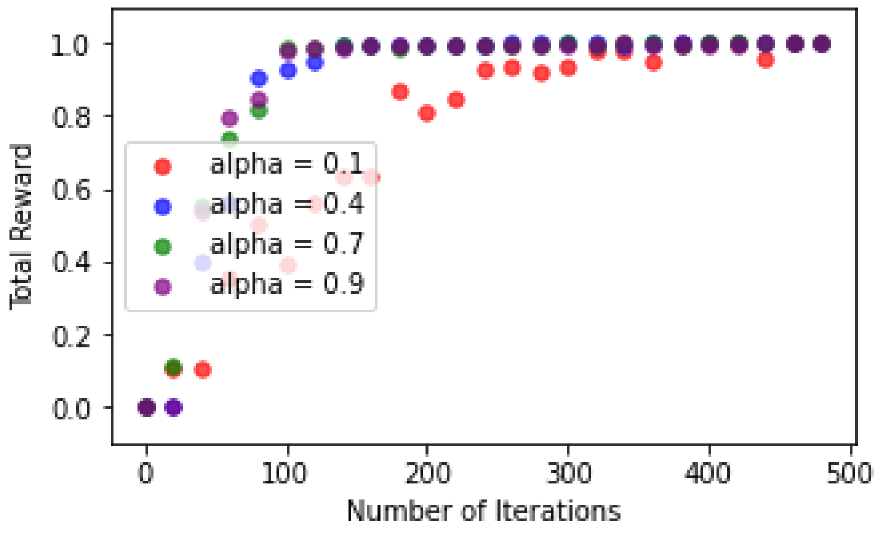
- **Discount factor**:



- **epsilon greedy**:



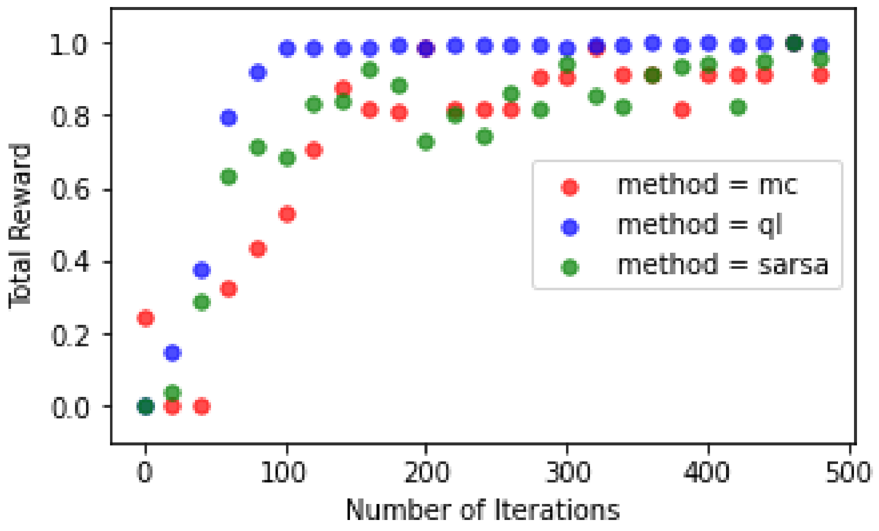
- **Alpha**:



we'll use gamma = 0.99, alpha = 0.9, epsilon = 0.99 and decay of (1-0.99).

3.3 Exercise 3

Here we should have solved a maze of 25x25 using the fastest method we found we tested the 3 models one against each other upon 25 runs:



We decided to demonstrate the best model free in comparison to model based (PI). We use QL since we saw it has the fastest convergence among the others. The videos of both are in our colab notebook below.

4 Discussion

In this work we tested 4 methods of RL to solve a maze: 1 of them is model-based, and 3 are model-free. 2 points that we want to point at are:

convergence time: it seem that PI is the method with the fastest convergence time, though it demands a complete knowledge of the model. it happens since there is no need for an agent to wandering around the maze and learn it, but a simple mathematical calculations are enough. The advantage of model-free methods is when the calculations are no longer simple but getting more and more complex and its not easy to have a whole knowledge of the model.

start point: We saw that when training MC model without using random start point but always starting from (0,0), doesn't work well. So we used random start points, but to make sure it can start well in runtime, and also for the sake of measurement we started from (0,0) each 1/20 runnings. This, unlike QL and SARSA where it works well even when starting always from (0,0). It happens due to the difficult of MC to learn since it updates only once in episode.

5 Code

google colab notebook with our code