Problem 1: The Maze and the Random Minotaur

Guanyu Lin
19980514-5035
guanyul@kth.se

Tawsiful Islam
20001110-2035
tawsiful@kth.se

1 Basic maze

(a) MDP formulation for simultaneous moves

For the MDP of this problem, we will define the state space, action space, transition probabilities, rewards, and the finite-time horizon objective.

The states s in the state space consist of two tuples with the player's coordinate position (i_p, j_p) and the minotaur's position (i_m, j_m). The player needs to be within the maze's boundary and not be in a position where the walls are. The minotaur can go through the inner walls of the maze. The state space is thus defined as

$$\mathcal{S} = \big\{s = ((i_p, j_p), (i_m, j_m)) : (i_p, j_p) \neq \text{a wall}, (i_m, j_m) \neq \text{boundary wall}\big\}. \tag{1}$$

The boundary wall here considers the outer walls the minotaur cannot go through. The minotaur can walk through the walls inside the maze, which the player cannot do. These inner walls and the boundary walls are included in $(i_p, j_p) \neq$ a wall. There are two terminal states. One is where $(i_p, j_p) = (i_m, j_m)$ in s, where the player is eaten, and the second is $(i_p, j_p) =$ goal, where the player has reached the goal.

The action space is

$$\mathcal{A} = \{\text{up, down, left, right, stay}\}. \tag{2}$$

The minotaur's random walk is embedded into the transition probabilities.

The rewards are

$$r(s = \mathcal{S}_{(i_p,j_p)=(i_m,j_m)},\ a = \text{any}) = -\infty \text{ ; player is eaten} \tag{3}$$
$$r(s \neq \mathcal{S},\ a = \text{invalid move}) = -\infty \text{ ; colliding with wall} \tag{4}$$
$$r(s = \mathcal{S},\ a = \text{any}) = -1 \text{ ; taking one step} \tag{5}$$
$$r(s = \mathcal{S}_{(i_p,j_p)=\text{goal}},\ a = \text{any}) = 0 \text{ ; player reaching goal.} \tag{6}$$

For (5), this considers states when a player makes a move where it would collide with the wall

The transition probabilities are

$$P(s' = S | s = S, a = \text{valid move}) = \frac{1}{N_a} \text{ ; } N_a = \text{\#actions the minotaur can take} \tag{7}$$
$$P(s' \neq S | s = S, a = \text{invalid move}) = 0 \tag{8}$$
$$P(s' = S_{(i_p,j_p)=(i_m,j_m)} | s = S_{(i_p,j_p)=(i_m,j_m)}, a = \text{any}) = 1 \tag{9}$$
$$P(s' = \mathcal{S}_{(i_p,j_p)=\text{goal}} | s = \mathcal{S}_{(i_p,j_p)=\text{goal}}, a = \text{any}) = 1. \tag{10}$$

s' is the next state and is dependent on time. Here you have (7) that considers a step to an empty cell. The probability is $\frac{1}{N_a}$ because the minotaur follows a random walk even if we have a deterministic transition for our own state. If you are near the minotaur, it's still the same probability as (7) to get eaten at the next state where both meet in the same cell. (8) considers actions the player and minotaur cannot take where they collide with a wall they cannot go through. (9) and (10) are terminal states.

The objective for the finite-time horizon is

$$\max_{\pi} \mathbb{E}\Big\{\sum_{t=0}^{T} r_t^{\pi}(s_t, a_t)\Big\} \tag{11}$$

(b) Alternating Move

If we would allow the player and the minotaur to alternate their play and not play simultaneously, most things that have been mentioned above would be the same. The difference would be for the transition probabilities when they take a valid move. The probabilities would be

$$P(s' = ((i_p', j_p'), (i_m, j_m))|s = ((i_p, j_p), (i_m, j_m)), a = \text{valid move}) = 1 \tag{12}$$

$$P(s' = ((i_p, j_p), (i_m, j_m))|s = ((i_p, j_p), (i_m, j_m)), a = \text{stay}) = 1 \tag{13}$$

$$P(s' = ((i_p, j_p), (i_m', j_m'))|s = ((i_p, j_p), (i_m, j_m)), a = \text{valid move}) = \frac{1}{N_a} \tag{14}$$

where the player's move is deterministic, but the minotaur will do a random walk as long as the actions it takes are valid and do not collide with the boundary wall.

2 Dynamic Programming

(c) Shortest path T=20

The figure below illustrates this policy through text, where, for instance, "Player0" denotes the player's position at round 0, and "Player1" denotes the player's position at round 1.

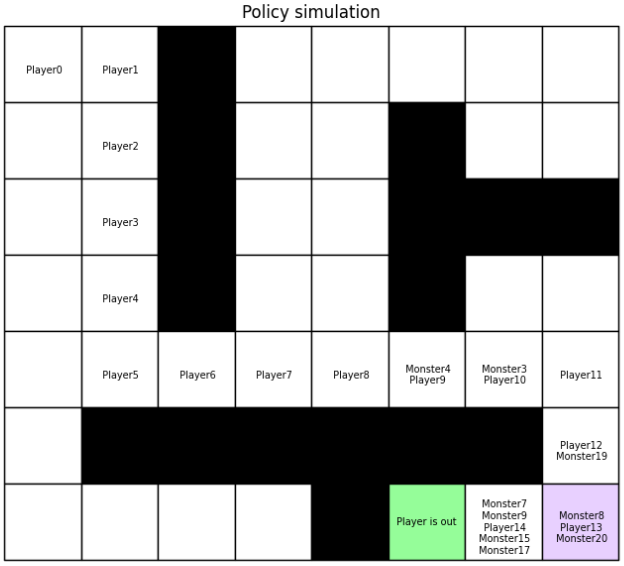

Figure 1: The figure shows the optimal path the player takes to reach the goal and what moves the minotaur is taking.

(d) Shortest path for different time horizons T and allow/not allow minotaur to stay

We conducted simulations under two scenarios: one where the minotaur is allowed to stay and another where it is not allowed to stay. We repeated these simulations 10,000 times for each value of T ranging from 1 to 30. The objective was to determine if, by the end of the simulation, the player occupied the exit grid. A simulation was deemed successful if the player reached the exit without encountering the minotaur in the same grid at any point during the episode.

Notably, when T exceeded 15 in both scenarios, success rates began to rise. This is consistent with the expectation that it would take at least that many steps for the player to reach the exit.

Comparing scenarios with and without allowing the minotaur to stay, a discernible pattern emerged: allowing the minotaur to stay resulted in a significantly lower success rate. This observation may be attributed to the fact that when the minotaur is permitted to stay, there is a higher likelihood of it remaining in the vicinity of the exit due to its initial spawn point at the exit.

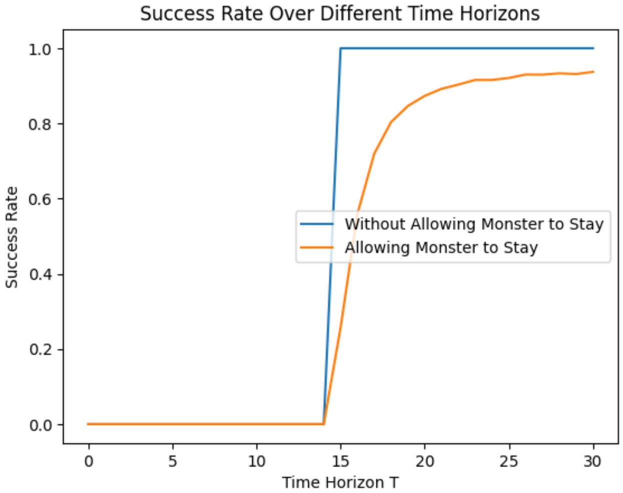

Figure 2: The y-axis shows the success rate the player exits the maze alive for different time horizons on x-axis for cases when the minotaur is allowed and not allowed to stand still.

3 Value Iteration

(e) Exiting maze in poisoned state

We choose to formulate the problem as an MDP with an infinite time horizon. Our objective will thus change to

$$\max_{\pi} \mathbb{E}\Big\{\sum_{t=0}^{\infty} \lambda^t r_t^{\pi}(s_t, a_t)\Big\}. \tag{15}$$

Knowing that the player's life is geometrically distributed with mean T=30 timesteps, we can calculate the discount factor to be

$$\mathbb{E}[T] = \frac{1}{1-\lambda} = 30 \Rightarrow \lambda = \frac{29}{30}. \tag{16}$$

(f) Exiting maze after simulating 10 000 games

When implementing the code, we add a condition in every iteration that random.random() $< \frac{29}{30}$ to essentially check if the generated random value falls below the threshold, simulating the geometric distribution of life with a mean of 30.

The probability of getting out alive using this policy by simulating 10,000 games is **62.01%**.

4 Additional Questions

(g) Theoretical questions:

1) What does it mean that a learning method is on-policy or off-policy?

- **On-policy learning methods:** These methods update the policy that is currently being used to make decisions. As the agent interacts with the environment, it updates its policy based on the outcomes of its actions. This means that the learning and decision-making processes are intertwined, and the policy that the agent uses to explore the environment is the same one that it is trying to improve.
- **Off-policy learning methods:** On the other hand, off-policy learning methods learn about an optimal policy independently of the agent's actions. They use data collected from a different policy to update their knowledge about the optimal policy. This means that the policy used for learning (the target policy) can be different from the policy used for decision-making (the behaviour policy). An example of an off-policy learner is Q-learning.

2) State the convergence conditions for Q-learning and SARSA. For Q-learning:

- The learning rates must approach zero, but not too quickly. Formally, this requires that the sum of the learning rates must diverge, but the sum of their squares must converge.
- Each state-action pair must be visited infinitely often. This has a precise mathematical definition: each action must have a non-zero probability of being selected by the policy in every state, i.e., $\pi(s, a) > 0$ for all (s, a). In practice, using an ε-greedy policy (where $\varepsilon > 0$) ensures that this condition is satisfied.

For SARSA:

- The learning rate parameter α must satisfy the conditions: $\sum \alpha_{nk}(s, a) = \infty$ and $\sum \alpha_{nk}^2(s, a) < \infty$ for all $s \in S$, where $n_k(s, a)$ denotes the k-th time (s, a) is visited.
- The exploration parameter ε (of the ε-greedy policy) must be decayed so that the policy converges to a greedy policy.
- Every state-action pair is visited infinitely many times.

(h) MDP formulation with key

To modify the Markov Decision Process (MDP) for the new scenario, we would need to adjust the state space, transition probabilities, and reward function.

State space $\mathcal{S}$

The state space would now need to include the information on the keys. So, a state could be defined as a

$$\mathcal{S} = \{s = ((i_p, j_p), (i_m, j_m), k) : (i_p, j_p) \neq \text{a wall}, (i_m, j_m) \neq \text{boundary wall}, k \in (0, 1)\} . \quad (17)$$

There is an additional term k that represents the information of the key. For $k = 0$ means that the player doesn't have the key, for $k = 1$ means that the player has visited the key position and has the key.

Action space $\mathcal{A}$

The action space remains the same, where the player and minotaur can move into neighbouring blocks but not diagonally.

Transition probabilities $\mathcal{P}$

The transition probabilities would need to be updated to reflect the new behaviour of the minotaur. Now, with a 35% chance, the minotaur moves towards the player, and with a 65% chance, it moves uniformly at random in all directions.

$P(s' = S|s = S, a = \text{valid move}) = \dfrac{1}{N_a}$
$P(s' \neq S|s = S, a = \text{invalid move}) = 0$
$P(s' = S_{(i_p,j_p)=(i_m,j_m)}|s = S_{(i_p,j_p)=(i_m,j_m)}, a = \text{any}) = 1$; terminal state (minotaur eats player)
$P(s' = \mathcal{S}_{(i_p,j_p)=\text{goal}}|s = \mathcal{S}_{(i_p,j_p)=\text{goal}}, a = \text{any}) = 1$; terminal state (player reaches goal)
$P(s' = S|s = S, a = \text{minotaur moves towards player}) = 0.35$
$P(s' = S'|s = S, a = \text{minotaur moves randomly}) = \dfrac{0.65}{N_a}$
$P(s' = \mathcal{S}_{(i_p,j_p)=\text{key pos},k=1}|s = \mathcal{S}_{(i_p,j_p)\neq\text{key pos},k=0}, a = \text{move to the key}) = 1$; player gets key

Rewards $\mathcal{R}$

The rewards are

$$r(s = \mathcal{S}_{(i_p,j_p)=(i_m,j_m)},\ a = \text{any}) = -1 \qquad \text{; player is eaten} \tag{18}$$
$$r(s \neq \mathcal{S},\ a = \text{invalid move}) = -1 \qquad \text{; colliding with wall} \tag{19}$$
$$r(s = \mathcal{S},\ a = \text{any, player doesn't have key}) = -0.1 \qquad \text{; taking one step} \tag{20}$$
$$r(s = \mathcal{S},\ a = \text{any, player has key}) = 0 \qquad \text{; taking one step} \tag{21}$$
$$r(s = \mathcal{S}_{(i_p,j_p)=\text{goal}},\ a = \text{any}) = 1 \qquad \text{; player reaching goal.} \tag{22}$$
$$r(s = \mathcal{S}_{(i_p,j_p)=\text{key}},\ a = \text{any}) = 0.5 \qquad \text{; player gets the key.} \tag{23}$$

Objective

The objective is

$$\max_{\pi} \mathbb{E}\left\{ \sum_{t=0}^{\infty} \lambda^t r_t^{\pi}(s_t, a_t) \right\}. \tag{24}$$

with $\lambda = 49/50$ as the expected life is 50 and we calculated similar to (16)

5 Q-Learning and Sarsa

(i) Q-learning

(i).1 Describe your implementation in pseudo code

```
Initialise the Q-values and all n(s,a) for each (s,a) as zeros
For each episode k from 1 to N
    Initialise the environment, start with state s=s0 & t=0
    while episode k is not finished or not reached exit
        if probability < epsilon,
            choose random valid action a at s according to
            uniform distribution
        else
            choose action a according to policy
        n(s,a) <-- n(s,a) + 1 which is the times state (s,a) been visited
        step <-- 1/n(s,a)**alpha
        observe s and r
        Q(s,a) <-- Q(s,a) + step * (r + lambda * max(Q(s',a')) - Q(s,a))
                where a' gives highest Q-value at s'
        t <-- t+1
        s <-- s'
```

(i).2 Q-learning for 2 different exploration values

The time horizon that has been used for all simulation hereafter with this new MDP has $T = 200$ and we run the algorithms for 50000 episodes.

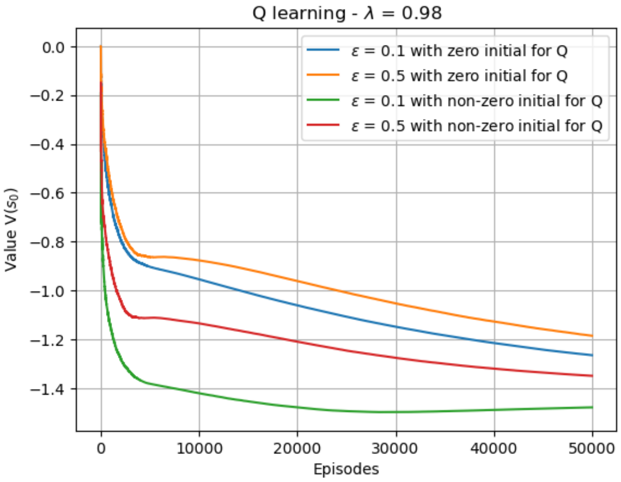

Figure 3: Convergence speed of different ϵ values and initialisation of Q-values for Q-learning.

From figure 3 we see two things that can be concluded. Regardless of initial Q-values, it can be seen that having lower ϵ helps with the convergence of the learning process. When there is a lower probability of randomness during the simulation, the expected value stays more consistent after a certain number of episodes, which is why one should have too high ϵ for exploration. It's clear that the figure doesn't show a clear convergence, but this is likely due to our combination of hyperparameters for the learning.

The second conclusion that can be drawn is that initialising the Q-values to be non-zero increases the speed of convergence. In theory, the Q-values should converge to the same value. However, initialising the Q-values as close as possible to the expected value, makes the learning faster and less number of episodes can be used. When comparing the different choices of ϵ and the choice of initialisation, the choice of initialisation should be prioritised for fast convergence.

(i).3 Q-learning for 2 different step sizes

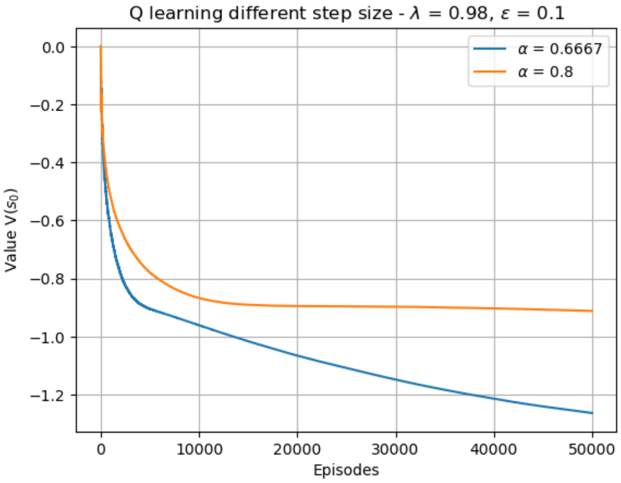

Figure 4: Convergence speed of different α values with zero initialisation of Q-values and $\epsilon = 0.1$.

Regarding the choice of exponent for step size, or learning rate in other terms, it's noticeable in figure 4 that a higher exponent helps with faster convergence as the step sizes reduce faster. On the other hand, the two lines do not converge to the same value. A plausible explanation is that the Q-learning converges too early causing there to be a bias in the policy. This is obviously a big problem where you want fast convergence, but low bias is necessary for a good policy.

(j) SARSA

(j).1 Difference between SARSA and Q-learning

The SARSA algorithm chooses a new action at each step, instead of choosing the optimal action at the next step as in Q-learning. In Q-learning, the action (a) for the next state is chosen based on the maximum Q-value for that state. This is a deterministic policy, where the agent selects the action that maximizes the Q-value.

In SARSA, both the current action (a) and the next action (a_{next}) are selected using an epsilon-greedy policy. The Q-value update is then based on the Q-value of the next state and the action taken in that state. SARSA uses an on-policy approach, where the Q-values are updated based on the actual policy being followed.

The initialisation for the Q-values will be zero for the simulations in the questions about SARSA.

(j).2 Plot of the value function over episodes of the initial states

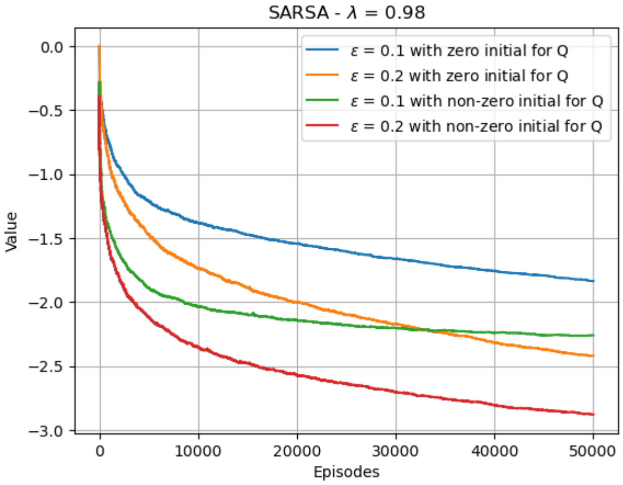

Figure 5: Convergence speed of different ϵ values and initialisation of Q-values for SARSA.

The plot 5 shows quite similar results as Q-learning when it comes to the initialisation of Q-value as mentioned earlier. The difference here with an on-policy method is that exploration is more important than the initialisation of the Q-values. A higher value on the ϵ gave a faster convergence. The plot also shows that the choice of ϵ should be prioritised over the initialisation of the Q-values. This is seen where $\epsilon = 0.2$ for zero initialisation converges faster than $\epsilon = 0.1$ for non-zero.

(j).3 Decaying exploration

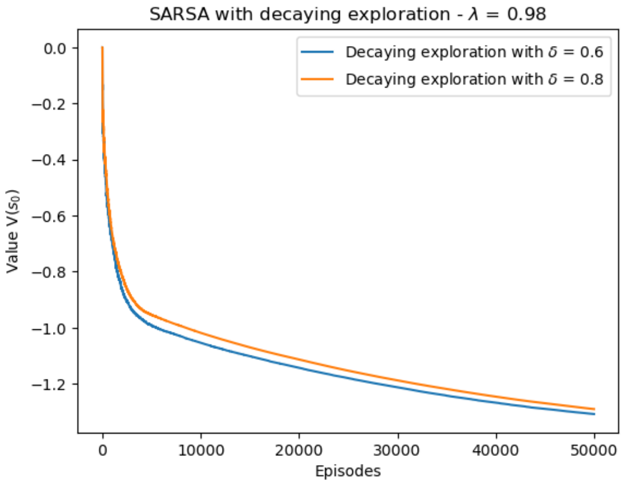

Figure 6: Convergence speed for different delta used for decaying exploration over the episodes.

Here we used decayed epsilon $\epsilon_k = 1/k^\delta$ with $\delta = 0.6$ and 0.8, to capture cases that $\alpha = 2/3 > \delta = 0.6$ and $\alpha = 2/3 < \delta = 0.8$. According to the plot, we can see that the convergence is slightly faster than a fixed epsilon with our parameters in the simulation. When $\alpha = 2/3 > \delta = 0.6$, the convergence speed is faster than $\alpha = 2/3 < \delta = 0.8$. Having $\delta > \alpha$ shows slower convergence as there is a slower learning rate and you allow more exploration before stabilising the policy. This is suitable for dynamic environments where you might need to account for changes later in time/later episodes. If you have a static environment $\delta < \alpha$, is reasonable as you can converge faster to a stable policy where there is less randomness. The risk you might face is overlooking valuable information during the simulation as the policy is less influenced by the environment.

(k) Probability of leaving the maze

To estimate the probability of exiting the maze we used the following hyperparameters seen in table 1:

Parameter	Value
Number of episodes	50 000
Time Horizon	300
α	$2/3$
λ	$49/50$
ϵ	0.1
δ	0.8

Table 1: Table contains the hyperparameters used to simulate 10000 games and estimate the probability of exiting the maze.

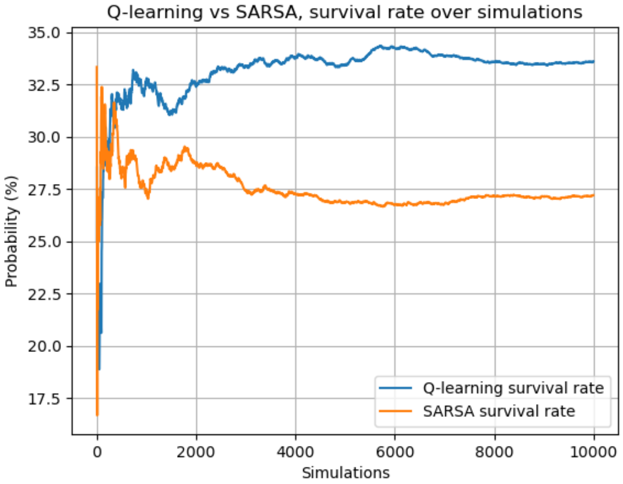

Figure 7: The probability in percentage for exiting the maze over 10000 simulations.

We simulated the respective policy we got from Q-learning and SARSA 10000 times. The Q-values were initialised with non-zero values. We got the survival rate is 33.59% for Q-learning and 27.19% for SARSA. The algorithms' convergence can be seen in figure 7. This is of course very dependent on our choice of hyperparameters and can be improved with different α, ϵ, δ, time horizon, and initial Q-values. Even the choice of rewards could improve the probability. In our case, the Q-learning was the better method to find a policy. Our probabilities are not equal to the initialised Q-values, which is due to how our reward function has been defined for the MDP. If the rewards would have been an indicator function for exiting the maze (1 for exiting and 0 for failing), then by definition the expected reward would have been the probability to exit the maze. This probability of surviving is also related to the definition of the value function $V(s_0)$ at initial state $s = s_0$.