KTH Royal Institute of Technology
Course: EL2805 - Reinforcement Learning

Saga Tran | 19991105 – 2182
Anh Do | 20020416 – 2317

Computer Lab 1

Date: 2025/12/05 18.00

Contents

1 Problem 1

1.1 Basic Maze

1.1.1 (a)

The problem is modeled as a finite horizon Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$.

1.1.1.1 State Space ($\mathcal{S}$)

The state space is defined by the joint position of the Player (P) and the Minotaur (M) within the grid.

$$\mathcal{S} = \{(p, m) \mid p \in \mathcal{C}, m \in \mathcal{D}\} \tag{1}$$

where $\mathcal{D}$ represents the set of all discrete cells in the maze grid, while $\mathcal{C}$ represents the set of all discrete non-wall cells in the maze grid (e.g., p and m have each coordinates (x, y) and $\mathcal{C} \subset \mathcal{D}$). The terminal states are:

- **Win State:** $s_{win} = \{(p, m) \in \mathcal{S} \mid p = B, m \neq B\}$. The player reaches exit B without being caught.
- **Loss State:** $s_{loss} = \{(p, m) \in \mathcal{S} \mid p = m\}$. The Minotaur occupies the same cell as the player (caught).

Once in a terminal state (e.g., s_{win} or s_{loss}) the game transitions to the absorbing state s_{done}. This is mainly for handling the accumulation of rewards at s_{win}.

1.1.1.2 Action Space ($\mathcal{A}$)

At each time step t, the player can choose to move to an adjacent cell or stay still.

$$\mathcal{A} = \{\text{Stay, Up, Down, Left, Right}\} \tag{2}$$

Note: Moves are constrained by the maze walls for the player.

1.1.1.3 Transition Probabilities ($\mathcal{P}$)

The state transitions from $s_t = (p_t, m_t)$ to $s_{t+1} = (p_{t+1}, m_{t+1})$ occur simultaneously. The transition probability factorizes as:

$$P(s_{t+1} \mid s_t, a_t) = P(p_{t+1} \mid p_t, a_t) \times P(m_{t+1} \mid m_t) \tag{3}$$

- **Player Dynamics:** Deterministic. p_{t+1} is the result of action a_t applied to p_t. If the move hits a wall or is invalid, $p_{t+1} = p_t$.
- **Minotaur Dynamics:** Random Walk. The Minotaur moves to any adjacent cell within the grid boundaries with uniform probability. It cannot stand still and ignores internal walls.

$$P(m_{t+1} \mid m_t) = \frac{1}{|N(m_t)|} \quad \text{if } m_{t+1} \in N(m_t) \tag{4}$$

where $N(m_t)$ is the set of valid neighbors (Up, Down, Left, Right) for the Minotaur at position m_t, restricted only by the exterior borders of the maze.

Observe that some transitions result in a winning or losing state based on the conditions defined in 1.1.1.1. In these cases, if an action leads to multiple winning outcomes, we must sum their probabilities.

To handle the transitions to the absorbing states correctly, we define:

$$P(s_{done} \mid s_t, a_t) = \begin{cases} 1 & \text{if } s_t \in s_{win} \cup s_{loss} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

1.1.1.4 Reward Function ($\mathcal{R}$)

To maximize the probability of exiting the maze, we define the reward to be 1 only upon being at the "Win" state, and 0 otherwise. This transforms the expected cumulative reward into the probability of success.

$$R(s_t, a_t) = \begin{cases} 1 & \text{if } s_t = s_{win} \text{ (Reached B, not eaten)} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

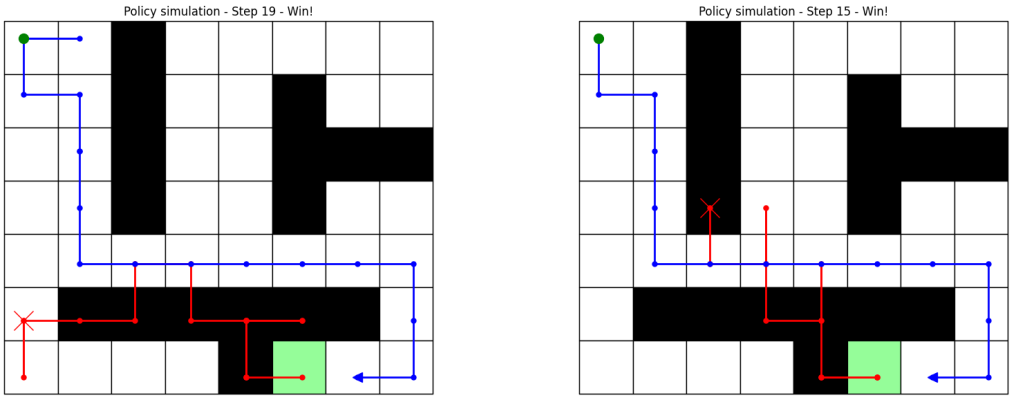
1.1.2 (b)

Yes, introducing the possibility for the minotaur to stand still makes the problem more difficult. This is because the agent must now account for an additional action of the minotaur, which increases the complexity of predicting its movements and planning a safe path to the goal. In the original problem, it is clear that we will never be eaten as long as we keep moving, due to the even steps between the player and the minotaur. However, allowing the minotaur to stand still breaks this parity, making it harder to find an optimal policy. In the extended problem, standing still could now become an optimal action in certain situations, further complicating the model.

1.2 Dynamic Programming

1.2.1 (c)

Using Dynamic Programming, we computed the optimal policy for the basic maze setup.



(a) With reward structure (1.1.1.4)

(b) With penalized stepping

Figure 1: Optimal policies for the basic maze using Dynamic Programming. The green circle indicates the starting position of the player, while the red cross marks the Minotaur's starting position. The blue triangle points to the exit point B. (a) Policy under the standard sparse reward. (b) Policy when a step penalty is introduced to discourage loitering.

Figure 1a shows that Dynamic Programming successfully finds the optimal policy to reach the exit. The model sometimes stops or moves randomly, which might look strange, but it makes sense given the 20-step horizon. The agent only needs 15 steps to exit and 16 steps to claim the reward, so the extra steps do not affect the total reward. The reward function in 1.1.1.4 depends only on reaching the exit and does not penalize time. Therefore, the policy can make unnecessary moves in the beginning as long as it reaches the goal safely.

In contrast, Figure 1b illustrates the optimal policy when we modify the reward function to penalize each step taken by the player. We observe that the agent now takes a more direct path to the exit, avoiding unnecessary moves and reaching the goal in exactly 15 steps.

1.2.2 (d)

Utilizing the reward structure defined in 1.1.1.4, the probability of escaping the maze is calculated directly from the value function at the start state at initial time ($t = 0$) that was obtained via Dynamic Programming.

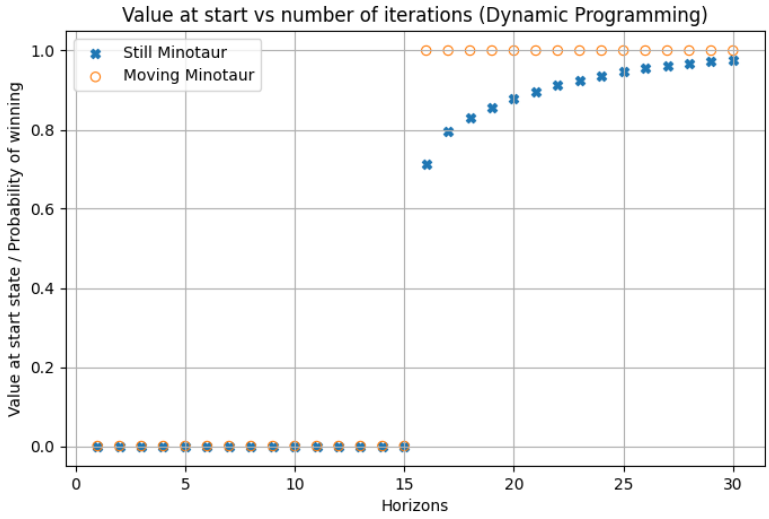


Figure 2: Probability of escaping the maze under different time horizons (T) using Dynamic Programming. The plot compares the scenarios where the Minotaur can and cannot stand still.

As shown in Figure 2, when the Minotaur cannot stand still, the escape probability reaches exactly 1 for time horizons $T \geq 16$. This sharp jump indicates that a deterministic winning policy exists. The strict movement constraints of the Minotaur allow the agent to exploit even step parity and guarantee an escape given sufficient time.

In contrast, when the Minotaur is allowed to stand still, the escape probability asymptotically approaches 1 but does not reach certainty within the observed finite horizons. Since the Minotaur retains the option to remain stationary, there always exists a non-zero probability that it will block the exit path for the duration of the episode, making a guaranteed escape impossible within a fixed time frame.

1.3 Value Iteration

1.3.1 (e)

We are given that the agent's life span L follows a geometric distribution with mean $\mu = 30$. The probability of dying at any time step t, denoted as p_{die}, is derived from the mean:

$$\mathbb{E}[L] = \frac{1}{p_{die}} = 30 \implies p_{die} = \frac{1}{30} \tag{7}$$

Consequently, the probability of surviving a single time step is $\gamma = 1 - p_{die} = \frac{29}{30}$.

To maximize the probability of exiting the maze alive, we must maximize the probability of reaching the goal state B at some time T multiplied by the probability of surviving the poison until time T. In the MDP framework, this survival probability acts identically to a discount factor. Therefore, we modify the MDP from problem 1.1.1 by introducing a discount factor $\gamma < 1$.

The tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ remains largely the same, with the following addition:

- **Discount Factor (γ):** We set $\gamma = \frac{29}{30} \approx 0.9667$.

Since the reward is only received once upon exiting (at time T), the expected return becomes $\mathbb{E}[\gamma^T \cdot 1]$. Since γ^T is exactly the probability of surviving T steps of poison, maximizing this value maximizes the probability of exiting alive. In this case the optimal $T^* = 15$. So the probability of exiting alive with the optimal policy would thus be $\gamma^{T^*} = \left(\frac{29}{30}\right)^{15} \approx 0.6014$.

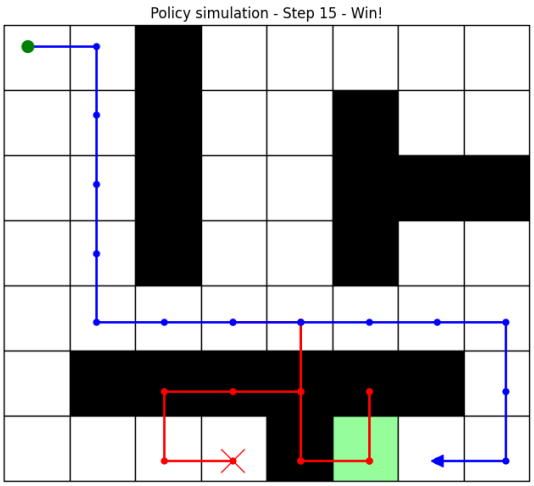


Figure 3: Optimal policy for the basic maze with poison using Value Iteration. The green circle indicates the starting position of the player, while the red cross marks the Minotaur's starting position. The blue triangle points to the exit point B.

1.3.2 (f)

1.4 Additional questions

1.4.1 (g)

1.4.1.1 On-policy vs. Off-policy Learning

The distinction between these learning methods lies in the relationship between the *behavior policy* (used to explore) and the *target policy* (being learned).

- **On-policy methods** (e.g., SARSA) estimate the value of the policy currently being used by the agent. The update rule depends on the action actually taken:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$$

  where a' is the action executed by the current policy, the same we used to get a.

- **Off-policy methods** (e.g., Q-learning) estimate the value of an optimal target policy independently of the agent's actions. The update rule uses the best possible

next action:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

This allows the agent to learn an optimal strategy even while exploring randomly, which means a' is chosen greedely instead and not by the current policy.

1.4.1.2 Convergence Conditions

For both algorithms to converge to Q^* with probability 1, the step sizes $\alpha_t(s,a)$ must satisfy the Robbins-Monro conditions:

$$\sum_{t=0}^{\infty} \alpha_t(s,a) = \infty \quad \text{and} \quad \sum_{t=0}^{\infty} \alpha_t^2(s,a) < \infty \tag{8}$$

Additionally, specific exploration conditions are required:

- **SARSA (On-policy)** Requires all state-action pairs must be visited infinitely often, and the policy must become greedy in the limit (e.g., $\epsilon \to 0$).
- **Q-learning (Off-policy)** Requires only that all state-action pairs are visited infinitely often. The behavior policy does not need to become greedy.

1.4.2 (h)

To accommodate the new requirements (keys, smarter Minotaur, and poison), we modify the MDP tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ as follows:

1.4.2.1 State Space Expansion

We introduce a binary variable $k \in \{0,1\}$ to track possession of the keys. The new state is $s = (p, m, k)$, where p is the player's position, m is the Minotaur's position, and $k = 1$ implies the key has been retrieved from C. We also need to modify the winning state to require the key aswell as not being eaten:

$$s_{win} = \{(p,m,k) \in \mathcal{S} \mid p = B, k = 1, m \neq B\} \tag{9}$$

1.4.2.2 Discount Factor (Poison)

To model the geometric life expectancy of 50 steps, we set the discount factor γ to the survival probability:

$$\gamma = 1 - \frac{1}{50} = \frac{49}{50} = 0.98 \tag{10}$$

1.4.2.3 Transition Probabilities

The transition $P(s_{t+1}|s_t, a_t)$ now accounts for the Minotaur's aggressive behavior. Let $\mathcal{N}(m_t)$ be the set of valid neighbors for the Minotaur. Let $\mathcal{N}^*(m_t, p_t) \subset \mathcal{N}(m_t)$ be the subset of neighbors that minimizes the Manhattan distance to the player p_t, which is given by:

$$m_{t+1} \in \mathcal{N}^*(m_t, p_t), \quad d(m_{t+1}, p_t) = \min_{m \in \mathcal{N}(m_t)} d(m, p_t) \tag{11}$$

The probability of the Minotaur moving to a specific neighbor $m' \in \mathcal{N}(m)$ is given by:

$$P(m_{t+1} \mid m_t, p_t) = \frac{0.65}{|\mathcal{N}(m_t)|} + \mathbb{I}(m_{t+1} \in \mathcal{N}^*(m_t, p_t)) \cdot \frac{0.35}{|\mathcal{N}^*(m_t, p_t)|} \tag{12}$$

Here, the first term represents the random behavior (uniform over all directions), and the second term adds probability mass to the optimal "chasing" moves. The key state k transitions to 1 deterministically as long as $s_t = (p_t = C, m, k = 0)$ and .

1.5 Q-Learning and Sarsa

1.5.1 (i) BOUNS

1.5.2 (j) BOUNS

1.5.3 (k) BOUNS

2 Problem 2

2.0.1 (a)

2.0.2 (b)

2.0.3 (c)

2.0.4 (d)

2.0.5 (e)

2.0.6 (f)