Reinforcement Learning: Lab 1

Martin Schuck, Damian Valle
`mschuck@kth.se, damianvs@kth.se`

1 Problem 1: The Minotaur maze

1.1 MDP problem formulation

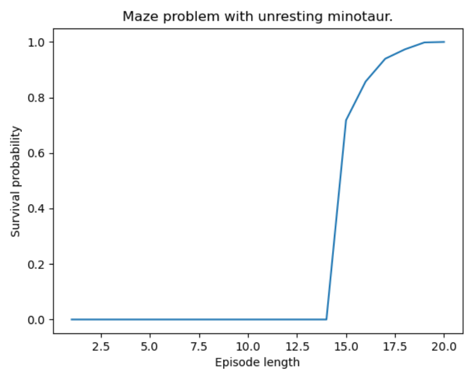
We propose the following MDP formulation:

- State space $\mathcal{S}$:
  We model the state space as the set of all possible positions of both the agent and the Minotaur inside the maze. This excludes any positions within the maze walls, as we assume the Minotaur to only be able to "break" through walls of thickness 1 within a single move. Formally, the state space is $\mathcal{S} = \big\{(i,j,k,l) \mid maze(i,j), maze(k,l) \neq 1\big\}$ where (i,j) and (k,l) are the coordinates of the player and the Minotaur respectively.
- Action space $\mathcal{A}$:
  We allow the agent to choose to either move `left`, `right`, `down`, `up` or not move at all (`stay`). Note that sometimes the player cannot move in a certain direction because of a wall, yet we permit this to be a valid action. We will see that this is not an issue as long as we define our transition probabilities and rewards appropriately. Formally, the action space is $\mathcal{A} = \{\texttt{up},\ \texttt{down},\ \texttt{left},\ \texttt{right},\ \texttt{stay}\}$.
- Transition probabilities $\mathcal{P}$:
  Note that there is no randomness involved upon taking an action by the player but the Minotaur follows a random walk throughout the maze. As a consequence, the transition probabilities are probabilistic. More precisely,
  - If at state s taking action a does not lead to a wall but to another valid position, then $\mathbb{P}(s'|s,a) = \frac{1}{|PossibleMinotaurActions|}$, where the set s' consists of the new player position and all possible new Minotaur positions.
  - If at state s taking action a leads to a wall or an obstacle, the player remains in his position and $\mathbb{P}(s'|s,a) = \frac{1}{|PossibleMinotaurActions|}$, where the set s' consists of the old player position instead, again combined with all possible new Minotaur positions.
- Rewards $\mathcal{R}$:
  The objective of the player is to minimize the time spent in the maze. Getting caught by the Minotaur freezes the player at the current position until the game ends. Therefore, the rewards are defined as follows:
  - If the agent makes a step at state s without reaching the goal or hitting any walls, it receives a reward of -1.
  - If the agent has reached the exit of the maze at state s (regardless of the Minotaur's position), it receives a reward of 0.
  - If the agent tries to take an impossible action (hits the maze's walls), it receives a reward of -100.
  - A remark on the Minotaur catching the agent: Due to the transition probabilities, no special reward handling is needed. The agent tries to make normal steps, but always stays at its position, earning a reward of -1 each round until the end of the game.

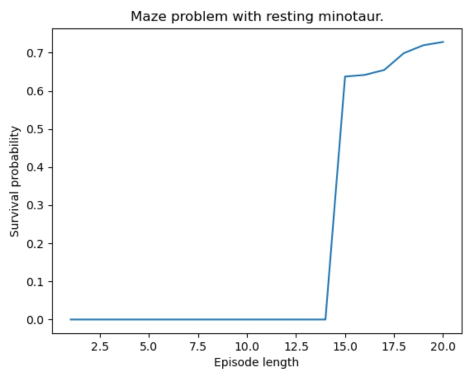
  Note: The rewards are independent of time (i.e. $r_t(.,.) = r(.,.)$).

1.2 Solving the Minotaur maze with DP

The problem can be solved with the standard dynamic programming approach, that is recursion from the final time step of the MDP to the beginning. With increasing time horizon, the agent's chances of exiting the maze increase as well. All horizons < 15 have a probability of 0 to succeed, as the shortest path to the exit is 15 steps long. As can be seen in figure 1, the agent always wins from $T = 20$ on. Allowing the Minotaur to stand still has a noticeable effect on the survival rate.



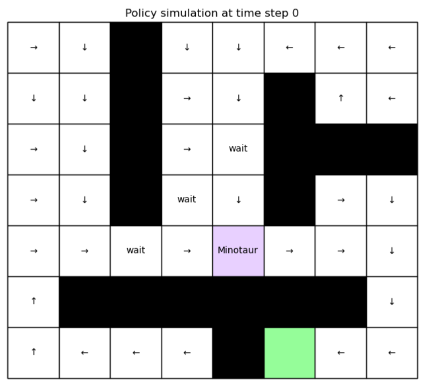
(a) Survival rate with Minotaur always on the move.



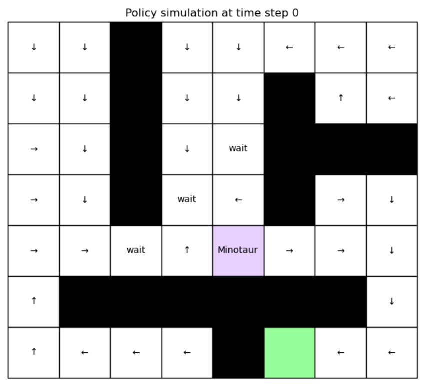
(b) Survival rate with Minotaur able to stand still.

Figure 1: Survival probability as a function of the time horizon T.

This can be explained as the agent can't be certain that the Minotaur will leave its position at the next step, so walking on its previous position is no longer a safe move. Figure 2 visualizes the change in strategy if the agent gets close to the Minotaur.



(a) Policy for always moving Minotaur.



(b) Policy for possibly standing Minotaur.

Figure 2: Visualization of the policy for a Minotaur standing at position (4,4). The strategy change is noticeable for positions close to the Minotaur, as in figure 2b, the previous Minotaur position is no longer a safe move.

1.3 Solving the Minotaur maze with geometric time horizons

As per the lecture, solving a MDP with geometrically distributed time horizon of expectation 30 is equivalent to solving the infinite horizon problem with discount factor $\lambda = \frac{29}{30}$. The infinite horizon problem also motivates a switch to value iteration instead of the standard recursion from the last time step from the previous solution. A converged strategy for this problem results in an approximate survival rate of 0.54 for a geometrically distributed time horizon with mean 30.

2 Problem 2: Robbing banks

2.1 MDP problem formulation

We propose the following MDP formulation:

- State space $\mathcal{S}$:

  We model the state space as the set of all possible positions of both the robber and the police inside the city. Formally, the state space is $\mathcal{S} = \big\{(i, j, k, l)\big\}$ where (i, j) and (k, l) are the coordinates of the robber and the police respectively.

- Action space $\mathcal{A}$:

  We allow the robber to choose to either move `left`, `right`, `down`, `up` or not move at all (`stay`). Note that sometimes the player cannot move in a certain direction because of a wall, yet we permit this to be action. We will see that this is not an issue as long as we define our transition probabilities and rewards appropriately. Formally, the action space is

  $\mathcal{A} = \{\texttt{up},\ \texttt{down},\ \texttt{left},\ \texttt{right},\ \texttt{stay}\}$.

- Transition probabilities $\mathcal{P}$:

  Note that there is no randomness involved upon taking an action by the player but the police follows a random walk toward the robber. As a consequence, the transition probabilities are probabilistic. More precisely,

  - If at state (or position) s taking action (or move) a does not lead to a wall but to another state (or position) s', then $\mathbb{P}(s'|s,a) = \frac{1}{|PossiblePoliceActions|}$, where the set s' consists of the new player position and all possible new Police positions.
  - If at state (or position) s taking action (or move) a leads to a wall or an obstacle, the player remains in his state (or position) s, then $\mathbb{P}(s'|s,a) = \frac{1}{|PossiblePoliceActions|}$, where the set s' consists of the old player position and all possible new Police positions.

  Note: Recall that for a fixed $s \in \mathcal{S}$ and $a \in \mathcal{A}$ we have $\sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a) = 1$, thus if for some $S' \subset \mathcal{S}$ we have $\sum_{s' \in \mathcal{S}'} \mathbb{P}(s'|s,a) = 1$, then for all $s'' \in \mathcal{S} \backslash \mathcal{S}'$ we have $\mathbb{P}(s''|s,a) = 0$,

- Rewards $\mathcal{R}$:

  The objective of the player is to maximize the average discounted reward by staying as much as possible robbing banks while not getting captured by the police.

  - If at state s, taking action a, leads to staying in a bank and the police is on a different cell, then $r(s,a) = 10$.
  - If at state s, taking action a, leads to you staying in the same cell as the police, then $r(s,a) = -50$.
  - If at state s, taking action a, leads you not staying in a bank and the police staying in another cell, then $r(s,a) = 0$.

  Note: Here the rewards are independent of time (i.e. $r_t(.,.) = r(.,.)$).

2.2 Value function evaluation as a function of λ

Evaluating the value function at the initial state we get Figure 3. It can be seen that, for low values of lamda, the value function approaches 10 which is the initial state reward as the robbers start in the bank position. As lamda approaches 1, the value function diverges to infinity, this is due to optimal play yielding infinite reward that, if not discounted, sums to infinity.

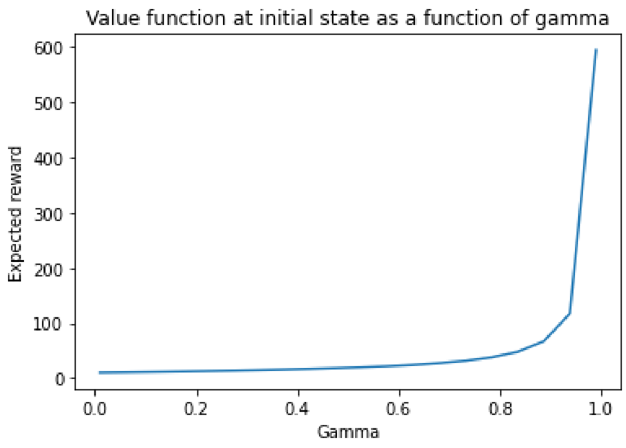
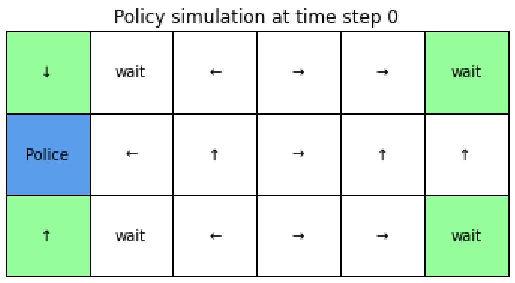
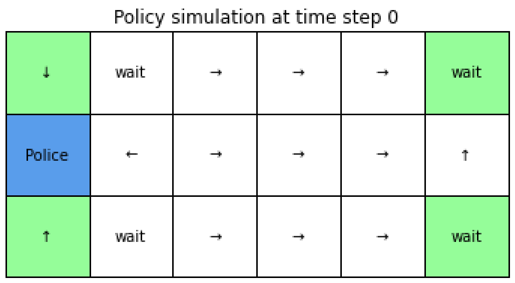


Figure 3: Value function at initial state as a function of lamda (stated as gamma in the plot but accounts for the discount factor)



(a) $\lambda = 0.01$



(b) $\lambda = 0.99$

Figure 4: Optimal policy visualization

2.3 Optimal policy as a function of λ

Looking at Figure 4a, it can be seen that low values of λ fail to understand long time rewards. For example, in cell $(1, 2)$ the agent chooses action up in order to quickly arrive to the closest bank while in Figure 4b the agent chooses to stay away from the police even if it means waiting for one more turn to reach the first reward.

3 Problem 3: Robbing banks (Reloaded)

3.1 Q-learning

We implement the Q-learning algorithm exploring actions uniformly at random. Figure 5 shows the evolution over 10 000 000 iterations of the value function. We see that the convergence has an offset related to the initial action. This is due to the start player square in $(0, 0)$, if `left` or `right` actions are chosen the initial reward is minus the infinite reward.

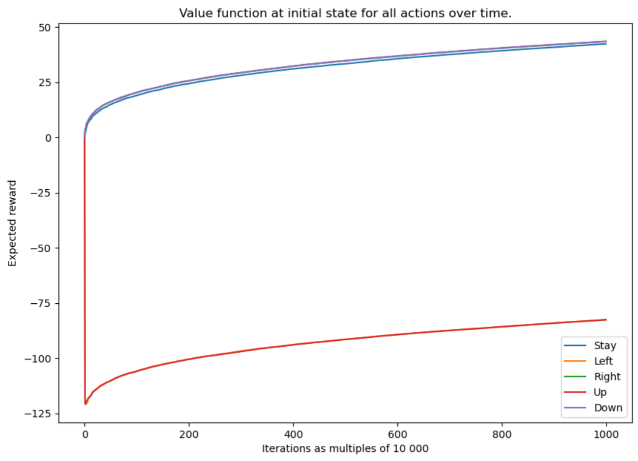


Figure 5: Value function vs time at initial state for each initial action.

3.2 SARSA

We implement the SARSA algorithm using ϵ-greedy exploration. Figure 6 shows the evolution of the algorith over time for different values of ϵ. It can be noted that very high values of ϵ yield smaller expected rewards since the policy behaves almost randomly (therefore stepping into the wall's infinite reward). Very small values of ϵ failed to converge due to insufficient iterations but we ran into computation time issues.

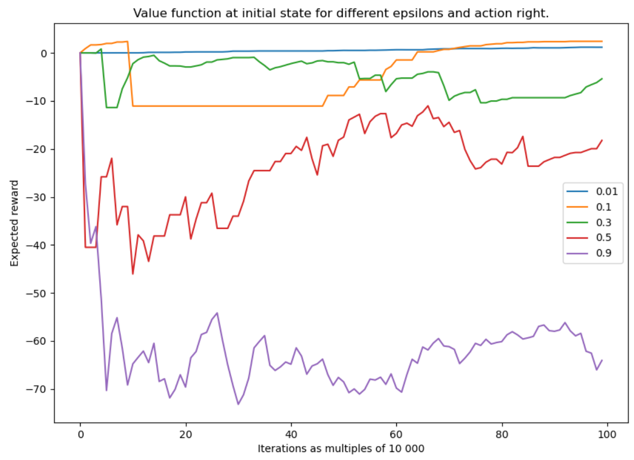


Figure 6: SARSA algorithm convergence for different ϵ

4 Problem 4: RL with linear function approximators

4.1 Training process

The algorithm tries to estimate the optimal Q-function of the problem without an environment model. As a basis for this Q-estimation, we employ a weighted Fourier basis of second order. It is defined as

$$\Phi_i(s) = cos(\pi \boldsymbol{\eta}_i^T s), \quad i \in \{1, ..., m\}$$
$$Q_w(s, a) = \boldsymbol{w}_a^T \Phi(s)$$

Sarsa(λ) then optimizes over the weights by acting on its current policy in the classical state $\rightarrow$ action $\rightarrow$ reward $\rightarrow$ state $\rightarrow$ action manner. To ensure sufficient exploration, we use an epsilon greedy policy which reduces exploration as training progresses. With the reward and the current Q estimates for s_t and s_{t+1}, weight updates are computed with Nesterov SGD in order to avoid oscillations in the updates. Since rewards are sparse in the mountain car problem, the updates are furthermore computed with eligibility traces that continuously perform increasingly discounted updates of previous actions' weights. Below, a complete list of all training parameters can be found.
Note: Even with good parameters, training is occasionally underperforming due to random weight initialization.

- Training episodes: 100
- Learning rate (α_i): $\alpha_i = 0.01/||\boldsymbol{\eta}||_2$. In addition to this, we reduce the learning rate by steps as we achieve better rewards.
- Eligibilit trace (λ): $\lambda = 0.75$
- Discount factor (γ): 1.0
- Fourier Basis: $\boldsymbol{\eta} = \left\{ \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \begin{pmatrix} 2 \\ 2 \end{pmatrix} \right\}$
- Weights: Initialized from a random Gaussian distribution with zero mean and 0.01 variance.
- Exploration rate (ϵ): We use a base exploration rate of $\epsilon = 0.3$ and it is reduced 9 times by 10% during the training process.
- Nesterov SGD: We use a momentum of $m = 0.1$.

The intuition behind the Fourier basis is that its terms should have a sensible meaning in the context of our state space. States are centered around $[0, 1]$, and the cosine maps all values to an interval of $[-1, 1]$. An η vector of $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ for example only considers the position of the car, and enables the Q-function to distinguish between positions close to the left and close to the right. $\begin{pmatrix} 2 \\ 0 \end{pmatrix}$, following the same logic, distinguishes only between states that are on the edges from those at the center, which is also a useful property for the Q-function.

4.2 Episodic reward during training

Figure 8 shows a high estimated value for high velocities and positions further to the right. Interestingly enough, positions further to the left also have high values since they are necessary in order to reach the flag.

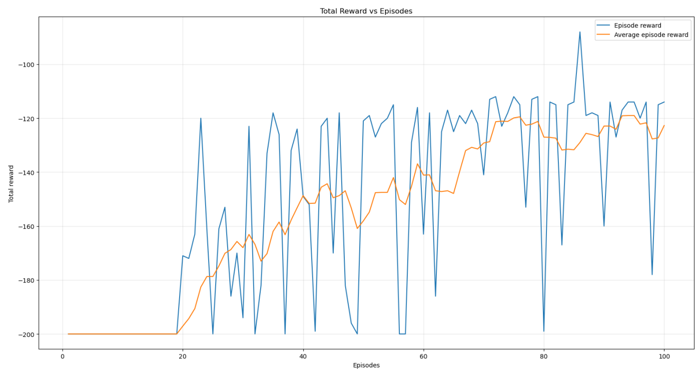


Figure 7: Episodic and average reward over 100 training episodes.

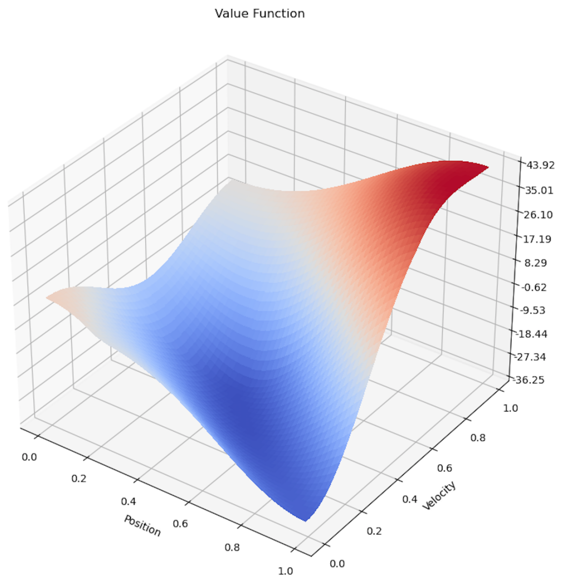
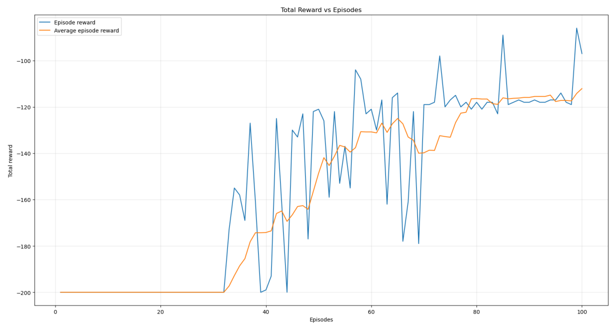


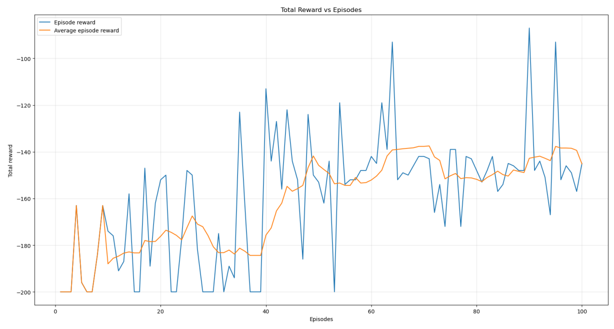
Figure 8: Value function of the estimated optimal policy.

4.3 Hyperparameter tuning: learning rate (α) and eligibility trace (λ)

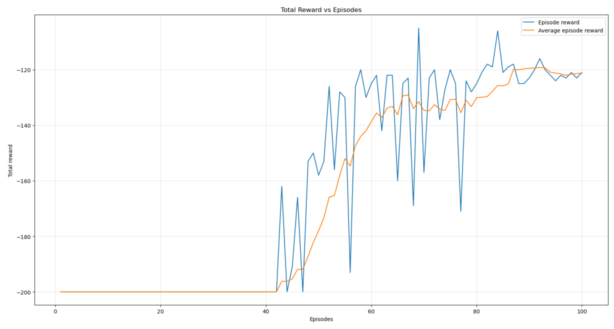
- **Low alpha** ($\alpha = 0.005$): The algorithm takes longer to learn a value function that reaches the flag.
- **High alpha** ($\alpha = 0.1$): The algorithm learns a way to reach the flag with less episodes but, due to large update steps, it fails to achieve steady improvements. Furthermore, we converge to a larger neighborhood of the optimal Q-function, which results in a policy that does not perform quite as well.
- **Low eligibility trace parameter** ($\lambda = 0.25$): Since the eligibility trace decays rapidly, weights for previously chosen actions are updated less. This explains the need for more episodes that are required to move the function towards a good estimate.
- **High eligibility trace parameter** ($\lambda = 0.97$): The algorithm fails to converge to a good solution and is highly dependent on different runs / weight initialization. Any eligibility trace parameter above 0.97 fails to converge at all.



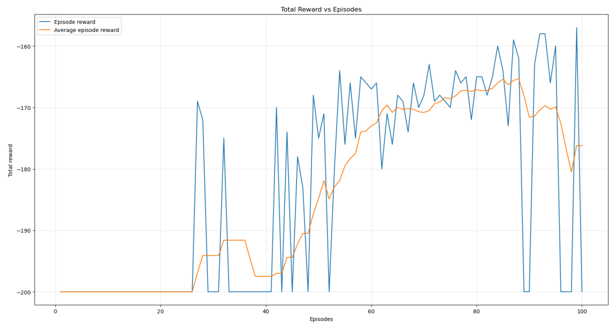
(a) Episodic reward over training episodes for low learning rate ($\alpha = 0.005$)



(b) Episodic reward over training episodes for high learning rate ($\alpha = 0.1$)



(c) Episodic reward over training episodes for low eligibility trace parameter ($\lambda = 0.25$)



(d) Episodic reward over training episodes for high eligibility trace parameter ($\lambda = 0.97$)