EL2805 Reinforcement Learning Computer Lab 1, 2022

Authors
Lea Keller - lmjke@kth.se - 19980209-4889
Jannik Wagner - wagne@kth.se - 19971213-1433

1 Problem 1 : Basic Maze

1.1 a)

The **states** are defined by the agent and the minotaur positions correlated at time t.

$$S = \{(\underbrace{(i,j)}_{s_a}, \underbrace{(k,l)}_{s_m}) : \text{where } (i,j) \text{ is not an obstacle}\}$$

Note that the minotaur can cross the obstacles but that the agent cannot. No moves are performed diagonally. The agent can take the following **actions**.

$$A = \{(\text{up}, \text{down}, \text{left}, \text{right}, \text{stay})\}$$

The agent is choosing where to move. The movements of the agent are deterministic. The minautor moves randomly in the environment. It chooses uniformly among the cells neighboring its current cell. Thus, there are 4 neighbors each with probability $\frac{1}{4}$ inside the maze 3 with probability $\frac{1}{3}$ at an edge and 2 with probability $\frac{1}{2}$ in a corner.

The transition probabilities can be defined in the following way:

- If in some state $s = ((i,j),(k,l))$ the Minotaur and Player position are the same, i.e., $(i,j) = (k,l)$, the player is eaten and thus the game ended and none will move anymore, then $\mathbb{P}(s|s,a) = 1$.
- If in some state $s = ((i,j),(k,l))$ the Minotaur and the player are not in the same position but the player is at the goal position, i.e., $B = (i,j) \neq (k,l)$, the player won and thus the game ended and noone will move anymore, then $\mathbb{P}(s|s,a) = 1$.
- If at state $s = ((i,j),(k,l))$ we are neither eaten nor at a goal state, i.e., $B \neq (i,j) \neq (k,l)$:
  - If taking action (or move) a does not lead to a wall or an obstacle, then the player position afterward will be $(i',j') \neq (i,j)$.
  - If, however, taking action (or move) a leads to a wall or an obstacle, the player remains in his position, i.e., $(i',j') = (i,j)$.
  - Additionally, let $(k'_h, l'_h), h = 1, ..., n$ be the n different positions to which the minotaur can move without moving into a wall or staying at the same position, but with moving into obstacles. Then $n = 2$ for corners, $n = 3$ for edges, and $n = 4$ else.
  - Then let $s'_h = ((i',j'),(k'_h,l'_h)), h = 1, ..., n$ be the different states we can end up in. Those will all have the same probability, i.e., $\mathbb{P}(s'_h|s,a) = \frac{1}{n}, h = 1, ..., n$.

The objective of the player is to find the exit of the maze while avoiding the obstacles.

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

The agent obtains different **rewards** regarding the different steps taken through the game.

The rewards can be defined in the following way:

- If at state s we are eaten, the reward for all actions a is $r(s, a) = 0$
- If at state s we are not eaten but at the exit, the reward for all actions a is $r(s, a) = 1$
- If at state s, taking action a, leads to a wall or an obstacle then $r(s, a) = 0$
- If at state s, taking action a, leads to some other position in the maze that is not the exit nor a wall nor an obstacle, then $r(s, a) = 0$.

Note that this reward design leads to the agent continuing to collect reward until the end of time when it reaches the goal state.

We also experimented with different rewards but found this simple reward design to be sufficient.

1.2 b)

Recall that the player can stand still but that the minautor needs to move.

- **The minautor and the player move one after another** : If the agent and the minotaur would end up next to each other and the minotaur were to move first, it could move to the agent's location and thus the agent would lose. However, if not restricted by obstacles, the agent would have needed to move towards the minotaur before that which can sometimes be avoided, but depending on obstacles it is not always the case. If the Player moves first, however, it can decide to move away from the minotaur if obstacles allow it. However, if it is trapped by the minotaur, it can not move away and thus will either move into the minotaur and be eaten or stay and be possibly eaten after the minotaur's move.
- **The minautor and the player moves simultaneously** : If the minotaur and the player end up next to each other, the player can always avoid being eaten by moving to the minotaur's location (if the minotaur is not on an obstacle) or to a free cell not reachable by the minotaur (if the minotaur is on an obstacle). When moving simultaneously to the minotaur's cell, they would cross each other without meeting at that cell.

The problem when the player and the minautor move at alternative rounds is modelled with the following changes to the previously defined MDP.

The difference is that when moving into the exit or into the minotaur, we will have reached an end state and thus the Minotaur won't take a step afterward.

More formally, if in state $s = ((i, j), (k, l))$ where $B \neq (i, j) \neq (k, l)$ (neither at exit nor eaten) and taking action a, where a is an allowed action that leads to $(i', j') \neq (i, j)$ with either moving into the minotaur $(i', j') = (k, l)$ or reaching the exit $B = (i', j')$, the minotaur will not take another move.

I.e., the probability of reaching state $s' = ((i', j'), (k, l))$ will be $\mathbb{P}(s'|s, a) = 1$ and the probability of reaching any other state $s'' \neq s'$ will be $\mathbb{P}(s''|s, a) = 0$.

Everything else is as before.

The different success probabilites over horizons $T = 1$ to $T = 20$ are shown in figure 1 together with the results of task 2.2.

The probability of being able to leave the maze when the player and the minotaur take turns seems to converge towards some value around 0.85.

2 Dynamic Programming

2.1 c)

A run of the computed policy can be seen in figure 2.

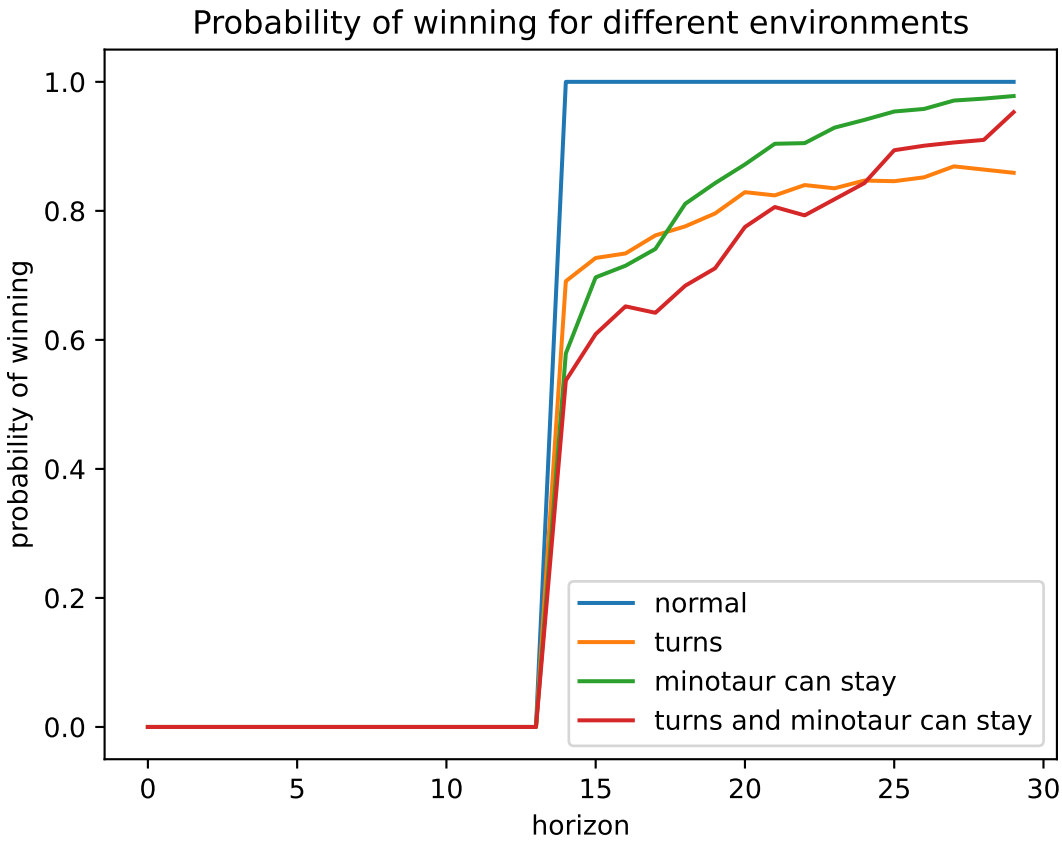


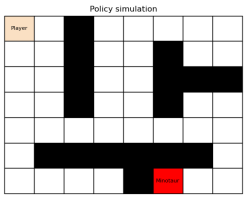
Figure 1: Success rates for different environments estimated with Monte-Carlo with 1000 episodes per horizon per environment.

2.2 d)

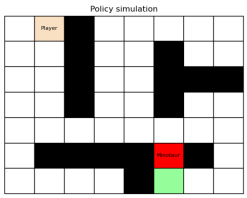
The success probabilities can be seen in figure 1.

If the minotaur is allowed to stand still, the probability to leave the maze is actually lower, as can be seen in figure 1. The reason might be that, as mentioned in section 1.2, if the game is simultaneous and the minotaur can't stand still, if the player and the minotaur stand next to each other, the player can always avoid being eaten by moving to the minotaur's position or, if the minotaur stands on an obstacle, by moving to a free cell different from the minotaur's position. Let's consider the case where the player is trapped by the minotaur in the case where the minotaur can stand still now. In this case, the player can't safely use the strategy of moving to the minotaur's position because the minotaur could decide to stay still and thus eat the player once the player has moved into the minotaur.

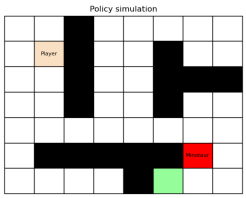
The probability of being able to leave the maze when the minotaur can stay seems to converge to 1. Possibly because given a high enough time horizon, the player can avoid getting stuck and possibly being eaten as described before.



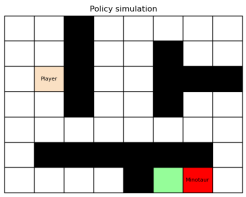
(a) $t = 0$



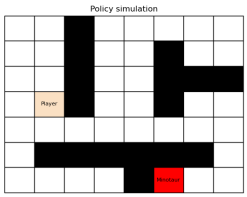
(b) $t = 1$



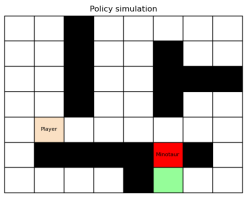
(c) $t = 2$



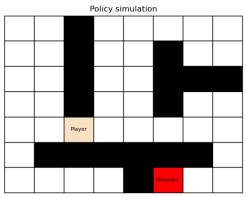
(d) $t = 3$



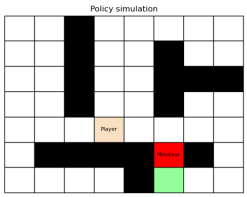
(e) $t = 4$



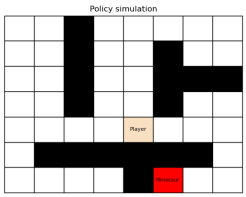
(f) $t = 5$



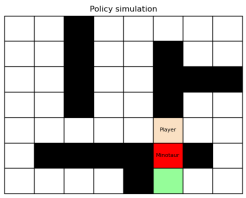
(g) $t = 6$



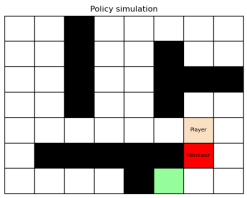
(h) $t = 7$



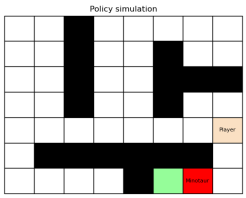
(i) $t = 8$



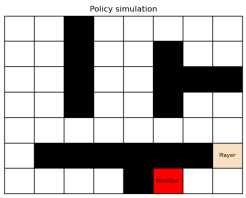
(j) $t = 9$



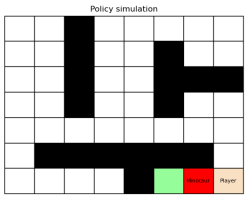
(k) $t = 10$



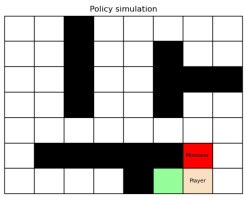
(l) $t = 11$



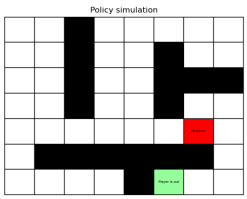
(m) $t = 12$



(n) $t = 13$



(o) $t = 14$



(p) $t = 15$

Figure 2: Simulation with a policy computed with dynamic programming for the environment described in 1.1.

	Exp1	Exp2	Exp3
p_{win}	0.5986	0.5926	0.5976
$V(((0,0),(6,5)))$ (start state)	18.0074	10.8294	111.2559
$V(((6,5),(0,0)))$ (a winning state)	29.9659	29.9659	200

Figure 3: Estimated success rate and values of the start state and a winning state for different experiments.

3 Value Iteration

3.1 e)

A geometric distribution over the time horizon with mean 30 is equivalent to having at every time step a chance of $\frac{1}{30}$ of dying from poison. Our first intuition was to adapt the MDP according to that in the following way:

We model dying from poison as the minotaur being teleported to the current player position. Let m be the mean of the geometric distribution. Then $\theta = \frac{1}{m}$ is the probability to die from poison at each step. Let $p(s'|s,a)$ be the probability distribution from before. Construct a new probability distribution $q(s'|s,a)$ from it as follows:

- If at state s the player is either eaten or at the exit, let $q(s'|s,a) = p(s'|s,a)$ as before, for all actions a and states s'.
- Else, if at state $s = ((i,j),(k,l))$, let $s' = ((i,j),(i,j))$ be the state where the minotaur was just teleported to the player's position. Then, for all actions a let $q(s'|s,a) = (1-\theta)p(s'|s,a) + \theta$ and for all states $s'' \neq s'$ let $q(s''|s,a) = (1-\theta)p(s''|s,a)$.

We chose to model dying from poison as the minotaur being teleported to the curent player position because this way we don't have to add new states since we previously decided not have specific terminal states but just make every winning and losing state terminal.

This works, however, the discount factor λ in the context of discounted MDPs is actually enough to model this, as it can be interpreted as assigning probability λ to reaching the next state when computing the expected sum of rewards. For that, one should choose $\lambda = 1 - \frac{1}{m}$ for mean m. For the estimation of the success rate one can then use the environment with adjusted probabilities, or also simplify things by just throwing a dice with probability $\frac{1}{m}$ in each step in the simulation.

For Q-learning and SARSA in section 5.1, we actually had more success, when we trained on an environment without adjusted transition probabilities (but still tested on an environment with appropriate probabilities for dying from poison).

3.2 f)

We used value iteration with 3 different settings:

Exp1: Probability of dying from poison in each step is 0 and $\lambda = 1 - \frac{1}{m} = \frac{29}{30}$

Exp2: Probability of dying from poison in each step is $\frac{1}{m} = \frac{1}{30}$ and $\lambda = 1 - \frac{1}{m} = \frac{29}{30}$

Exp3: Probability of dying from poison in each step is $\frac{1}{m} = \frac{1}{30}$ and $\lambda = 0$

In all cases the evaluation of the success rate p_{win} was done in the environment with probability of dying from poison $\frac{1}{m} = \frac{1}{30}$.

The estimated success rate and the values of the start state and a winning state for all experiments can be seen in table 3

4 Additional Questions

4.1 g) Theoretical questions

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

Off-policy: An off-policy learner learns the value of the optimal policy independently of the agent's actions. The policy used by the agent is often referred to as the behavior policy, and denoted by π_b.

On-policy: An on-policy learner learns the value of the policy being carried out by the agent. The policy used by the agent is computed from the previous collected data. It is an active learning method as the gathered data is controlled.

2) State the convergence conditions for Q-Learning and SARSA.

For the algorithms Q-learning and SARSA to converge, the behavior policy needs to ensure that, in infinite time, all (state,action) pairs are visited infinitely often, i.e., the behaviour policy leads to an irreducible Markov chain. This can be achieved by using an ϵ-greedy policy.

Furthermore, in theory the step sizes need to be square summable but its sum needs to diverge, i.e. $\sum_{t=0}^{\infty} \alpha_t = \infty$ and $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$. This can be achieved with $\alpha_t = \frac{1}{t}$.

Q-learning converges towards the real Q function (the state action value function given optimal behavior which does not depend on the behavior policy) whereas SARSA is converging to the state action value function with respect to the behavior policy. This leads to a safer behavior of the SARSA algorithm as it takes the ϵ-greedy exploration of the behavior policy into account, compared to the Q-learning algorithm.

By the definitions from before, Q-learning can thus be classified as an off-policy algorithm whereas SARSA can be classified as an on-policy algorithm.

4.2 h)

The Minotaur moves towards player with some probability

For the movements of the minotaur towards the player with probability $\theta = 35\%$, the transition probabilities have to be adjusted in the following way: Let $p(s'|s,a)$ be the probability distribution from before. Construct a new probability distribution $q(s'|s,a)$ from it as follows:

- If at state s the player is either eaten or at the exit, let $q(s'|s,a) = p(s'|s,a)$ as before, for all actions a and states s'.
- For all the states $s = ((i,j),(k,l))$, in which we are neither eaten nor at the exit ($B \neq (i,j) \neq (k,l)$), and actions a, let s' be the state where the player has moved according to his action and the minotaur has moved towards him/her (in the simultaneous case this would probably be towards the position where the player was before he/she took his/her action). Then, let $q(s'|s,a) = (1-\theta)p(s'|s,a) + \theta$ and for all states $s'' \neq s'$ let $q(s''|s,a) = (1-\theta)p(s''|s,a)$.

The player needs keys

For the modified problem with keys at some position C, we can add a boolean to each state to signal whether the keys were already picked up. I.e., $S = S_{old} \times \{0,1\} = \{(s,b) : s \in S_{old}, b \in \{0,1\}\}$, where S_{old} is the state space from before.

The transition probabilities need to be updated such that in a state, the player will only have exited when he/she is at the exit but not eaten, but additionally he/she needs to have the keys. Additionally, the player needs to be able to get the keys, thus we need to transition from not having the key to having the key, when the player currently does not have the key but after taking its action lands on C.

More formally, let's construct $q((s',b')|(s,b),a)$ from $p(s'|s,a)$ as before. For state $(s,b) = (((i,j),(k,l)),b)$, action a and the following state $(s',b') = (((i',j'),(k',l')),b')$:

- if $b = b' = 1$, the situation is equivalent to the environment without key, thus $q((s', 1)|(s, 1), a) = p(s'|s, a)$
- if $b = 0$, also let $q((s', 0)|(s, 0), a) = p(s'|s, a)$ except for the following exceptions:
  - the player cannot win yet when moving onto the exit B, thus, if $B = (i, j) \neq (k, l)$, with action a and if (i', j') is the new position according to a and k', l' is one of n possible new positions of the minotaur, then $q((s', 0)|(s, 0), a) = \frac{1}{n}$ (and then adjust it according to the adjusted minotaur movements as before).
  - when going to C without having the key ($b = 0$), we pick up the key ($b' = 1$), thus, if $(i', j') = C$, then $q((s', 1)|(s, 0), a) = p(s'|s, a)$.

5 BONUS

5.1 i)

We decided to reduce the number of terminal states to one winning state representative $((6, 5), (0, 0), 1)$ and one losing state representative $((0, 0), (0, 0), 0)$. This was done by transitioning to the winning state representative whenever we would have transitioned to any winning state and by transitioning to the losing state representative whenever we would have transitioned to any losing state. We did this to investigate whether it would make learning easier, but it doesn't seem like that.

Note also, that we give the reward of 1 when we are in the winning state and take any action and not when transitioning to the winning state. We decided to do so because otherwise we would have needed our rewards to also depend on the next state for situations in which we move to the exit and the minotaur based on randomness either also moves there or doesn't, which would result in a loss and thus no reward or a win and thus positive reward based on the following state.

This seems to cause a scaling of the value function by $\frac{1}{1-\lambda}$ due to a geometric series on the goal reward induced by the update rule. However, the learned policy is not affected by this.

5.1.1 1) Pseudocode

The pseudocode can be seen in algorithm 0. The Q-learning algorithm takes as arguments the environment (the maze here), the discount factor λ, the exponent of the step size α, the exploration parameter ϵ, and the number of episodes n.

5.1.2 2)

The training was done on an environment without poison included in the transition probabilities to avoid wasting episodes to dying from poison and doing no updates at all. Instead, the poison was modeled by setting $\lambda = 1 - \frac{1}{50} = 0.98$.

The experiments were done with $\epsilon = 0.1$ and $\epsilon = 0.2$.The plot can be seen in figure 4. Note that the learned Q function does not converge to the real Q function in 50.000 episodes. As can be seen in the figure, $\epsilon = 0.1$ seems to work slightly better, but the difference is insignificant.

It was crucial to the success of the training to initialize the Q function estimate in the following way: All entries were set to a value c slightly above 0 (we chose $c = 1 - \lambda$) such that all actions would be explored more quickly. The reason this helped is that when an action a was chosen in state s that did not lead to reward, its $Q(s, a)$ estimate would necessarily be removed, resulting in a different action being explored the next time the agent explored s when acting greedily. This can be seen from the update rule, here examplified for the very first visit of (s_t, a_t):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \frac{1}{N(s_t, a_t)^\alpha}(r + \lambda \max_a(Q(s_{t+1}, a)) - Q[s_t, a_t]) = c + \frac{1}{1}(0 + \lambda c - c) = \lambda c < c$$

Furthermore, we initialized all entries in Q with taking action stay to $-c$, i.e., $Q(\cdot, 0) = -c$, because we thought staying was not useful for exploration. This detail, however, was probably not crucial.

Algorithm 1 QLearning

Require: $env, \lambda, \alpha, \epsilon, n$
  Let $Q \in \mathbb{R}^{S \times A}$ ▷ Initiaize Q
  $Q[:, 1:] = 1 - \lambda$ ▷ initialize to a small value above 0
  $Q[:, 0] = -(1 - \lambda)$
  Let $N \in \mathbb{N}^{S \times A}$ ▷ Initialize state action appearance counts
  $N[:, :] = 0$
  for $k = 0, ..., n - 1$ **do** ▷ For each episode
    s, done = env.reset() ▷ Reset environment
    while not done **do**
      if random() $< \epsilon$ **then** ▷ Choose action epsilon greedily
        $a =$ randint$(0, A)$ ▷ Choose uniformly
      else: ▷ Choose randomly among actions attaining the maximum value
        $a =$ random.choice$(\arg\max_b(Q[s, b]))$
      end if
      $N[s, a] += 1$
      s_{t+1}, r, done = env.step(s, a) ▷ Take action
      $Q[s, a] = Q[s, a] + 1/N[s, a]^{\alpha}(r + \gamma \max_b(Q[s_{t+1}, b]) - Q[s, a])$ ▷ Update Q
      $s = s_{t+1}$ ▷ Update state
    end while
  end for
  return Q

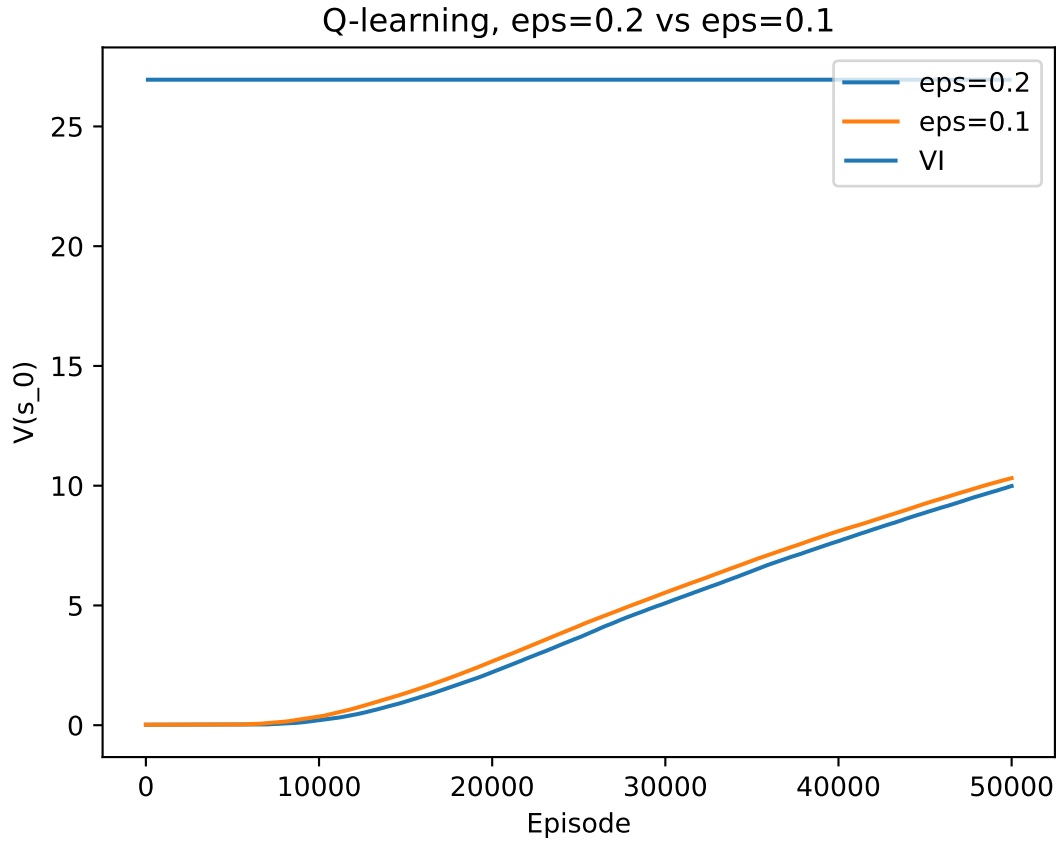


Figure 4: Q-learning with different values for ϵ, 50.000 episodes, $\lambda = 0.98$, $\alpha_t = \frac{1}{n_t(s,a)^{\frac{2}{3}}}$

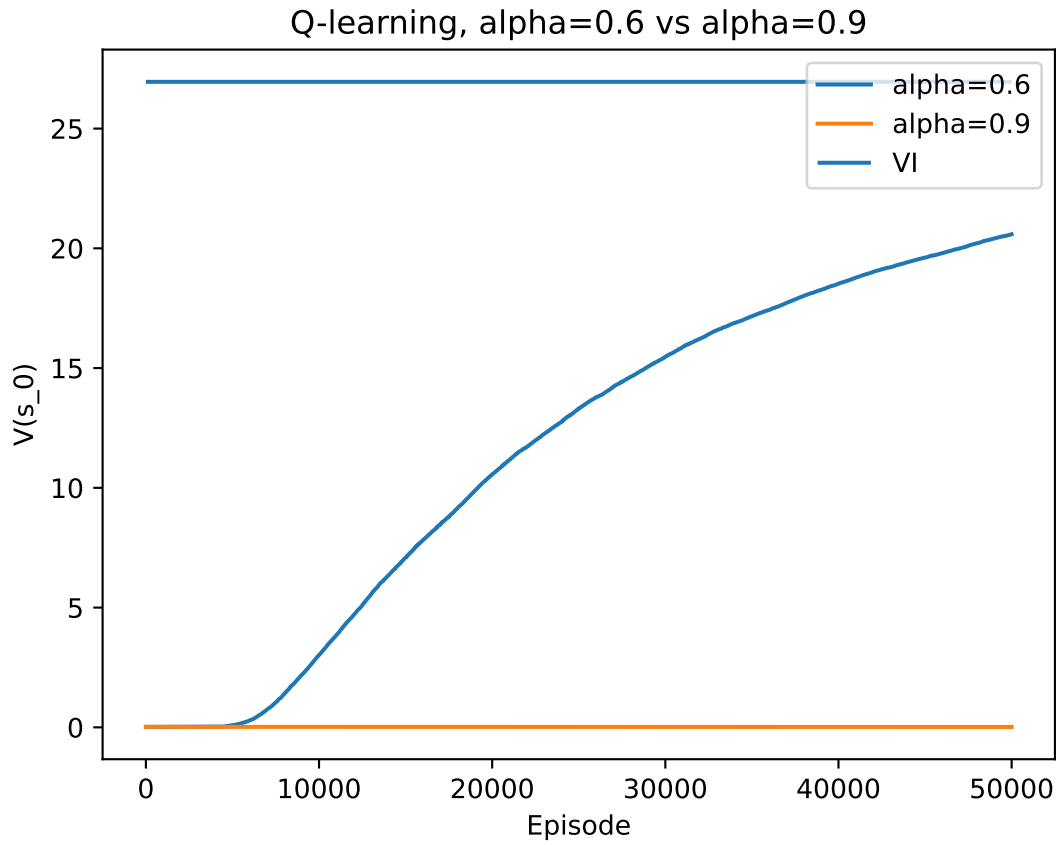


Figure 5: Q-learning with different values for α, 50.000 episodes, $\epsilon = 0.2$, $\lambda = 0.98$, $\alpha_t = \frac{1}{n_t(s,a)^\alpha}$

Finally, when acting greedily when there are several actions achieving $\max_b Q(s, b)$, we choose randomly among all actions achieving the maximum.

Additionally, of course, we also use an ϵ-greedy policy.

5.1.3 3)

We decided to fix $\epsilon = 0.2$. The experiments were done with $\alpha = 0.6$ and $\alpha = 0.9$. The plot can be seen in figure 5. Note that with $\alpha = 0.9$ the value of the starting state stays very close to 0 which looks like the algorithm does not learn at all. This is in fact not true, because the learned policy still receives an estimated success rate of 0.334. However, it is still quite remarkable that the value is so close to 0. This may be because with $\alpha = 0.9$ the step size $\alpha_t = \frac{1}{n(s,a)^\alpha}$ decays too quickly.

5.2 j) SARSA

5.2.1 1)

The main difference is in the update rule

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \frac{1}{N_t(s_t, a_t)^\alpha}(r + \lambda \max_a(Q(s_{t+1}, a)) - Q(s_t, a_t))$$

being replaced by

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \frac{1}{N_t(s_t, a_t)^\alpha}(r + \lambda(Q(s_{t+1}, a_{t+1})) - Q(s_t, a_t))$$

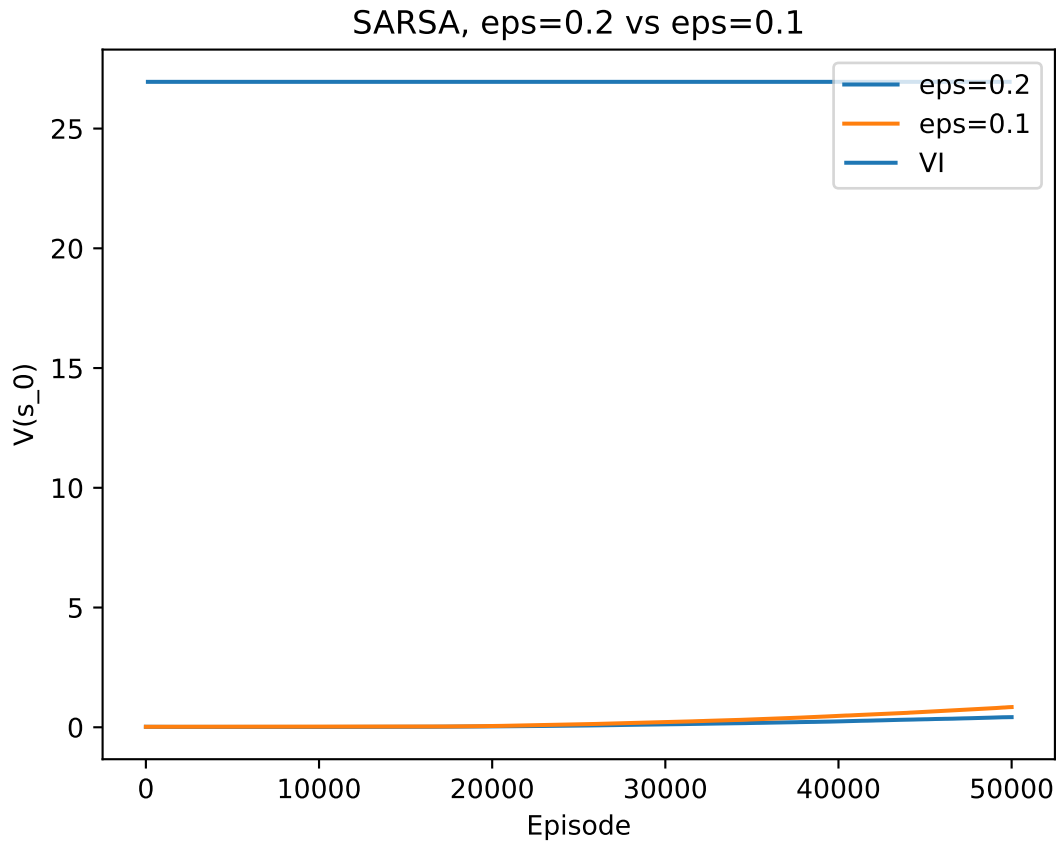


Figure 6: SARSA with different values for ϵ, 50.000 episodes, $\lambda = 0.98$, $\alpha_t = \frac{1}{n_t(s,a)^{\frac{2}{3}}}$

i.e., the target does not depend on assuming to behave optimal in the future but assuming to behave according to the policy in the future. For that reason, SARSA is on-policy learning and Q-learning off-policy learning. For this, we need to decide for the action in the next iteration a_{t+1} already in iteration t before doing the update in iteration t and remember it until iteration $t+1$. Furthermore, we add the possibility to let ϵ decay to 0.

5.2.2 2)

The plot can be seen in figure 6. Interestingly, the values are much lower than the real value or the values which Q-Learning converged too. This probably due to the fact that SARSA is an on-policy algorithm and we are using a non-optimal epsilon greedy policy under which the expected sum of rewards naturally is lower than optimal.

5.2.3 3)

We conduct experiments with $\delta = 0.6$ and $\delta = 0.9$. The results can be seen in 7. The performance is better with $\delta = 0.9$, thus, it seems to be better to have $\delta > \alpha$.

A consequence of $\delta > \alpha$ is that the step size decays slower than the exploration parameter, given that we reach each state action pair only once per episode. Because of that, the learning agent starts to behave near-optimal instead of making random decisions when its actions still have a considerable influence on the update of the Q-functions because the step size has not decayed that quickly.

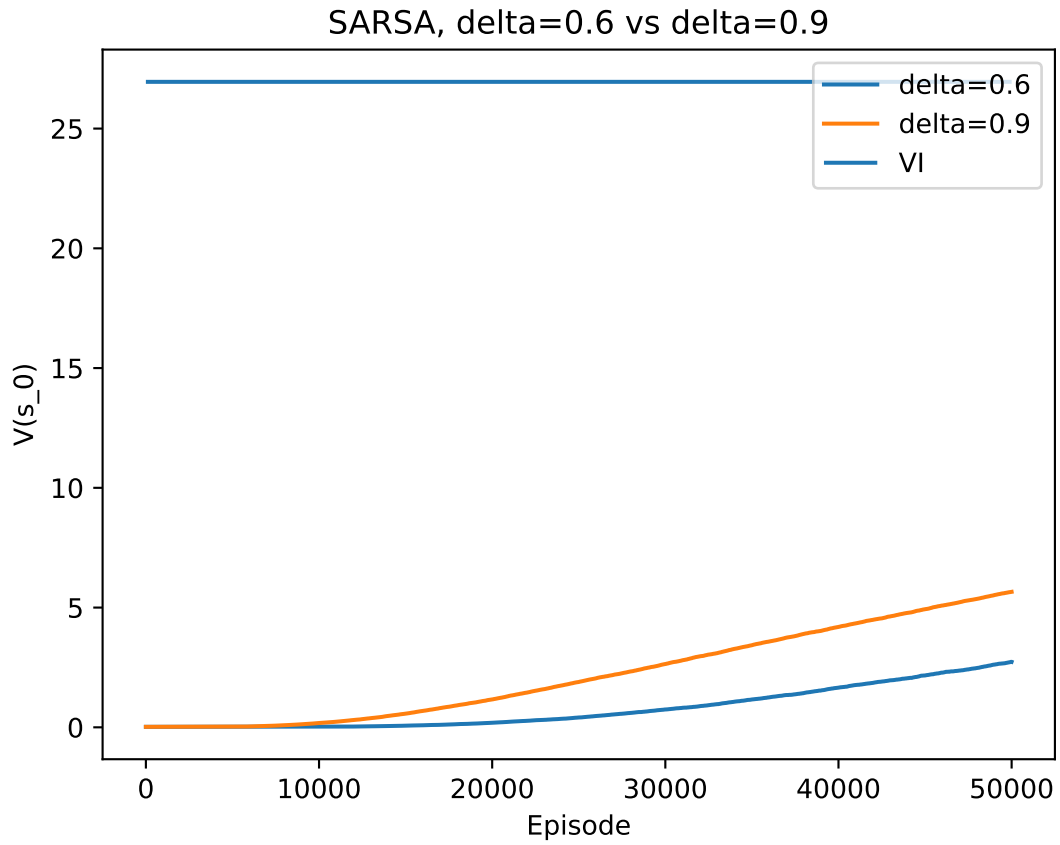


Figure 7: SARSA with different values for δ, 50.000 episodes, $\epsilon = \frac{1}{k^\delta}$, $\lambda = 0.98$, $\alpha_t = \frac{1}{n_t(s,a)^\alpha}$

6 k)

Success rate

We train and evaluate the values on an environment without the probability of dying from poison in the transition probabilities, but we of course test the success rate on an environment with this probability of dying from poison. Instead, the poison is modeled with λ during training as described before. For q learning, we chose the policy computed with $\epsilon = 0.2, \alpha = 0.6$ from section 5.1, referred to as QL3 here. For SARSA, we chose the policy with fixed $\epsilon = 0.2$ and $\alpha = \frac{2}{3}$, referred to as SARSA1 here. For comparison, we also consider the policy computed by value iteration, referred to as VI here. The results can be seen in table 8.

	VI	QL3	SARSA1
p_{win}	0.5565	0.5366	0.5538
$V(((0,0),(6,5),0))$ (start state)	26.9514	20.5893	0.4227
$V(((6,5),(0,0),1))$ (the winning state representative)	49.1206	48.4915	38.3803

Figure 8: Estimated success rate (using Monte Carlo over 10.000 episodes) and values of the start state and a winning state given value iteration, one experiment of QLearning and one experiment of SARSA.

As can be seen, the policy given by value iteration performs the best (as expected), very closely followed by the policy given by SARSA. The policy given by QLearning performs slightly less good but still relatively close. The fact that SARSA has a higher success rate than QLearning is in line

with the aforementioned property of SARSA usually learning safer because it takes into account the risks of exploration.

Interestingly, for the learned values of the start and winning state, qlearning follows value iteration very closely for the winning state and relatively closely for the starting state, whereas, SARSA has much lower values for the starting state and a bit lower values for the winning state. This is also in line with SARSA being an on-policy algorithm and the fact that we used a fixed ϵ, because SARSA learns the state action value with respect to our non optimal policy that is ϵ-greedy.

Q-value of initial state

As mentioned before, our agent keeps getting reward 1 until infinity once it reached the winning state. Thus we would expect it to have value $\frac{1}{1-\lambda} = 50$, for that reason we are a bit confused that value iteration converges to 49.1206 but we contribute it to numerical precision. Because of this scaling, our state values are not equal to probabilities, however, we consider it reasonable that without this scaling, i.e., without continuing to receive reward, the probability of winning would equal the value of the starting state in the undiscounted case:

$$V(s_{start}) = \mathbb{E}[\sum_{t=0}^{\infty} r(s_t, a_t)|s_0 = s_{start}] = 1 \times \Pr[win] + 0 \times \Pr[lose] = \Pr[win]$$

As mentioned previously, the poison probability can also be modeled using the discount factor, thus we assume that it should also be the case for

$$\mathbb{E}[\sum_{t=0}^{\infty} \lambda^t r(s_t, a_t)|s_0 = s_{start}]$$

when not having the poison probability in the transition probabilities.