EL2805 Reinforcement Learning Lab1

Panwei Hu 980709-T518, Mikael Westlund 980321-7851

January 10, 2022

1 Problem 1

1.1 1a

The problem is a finite horizon MDP with horizon T. The objective is thus to maximize $\mathbb{E}[\sum_{t=0}^{T} r_t(s_t, a_t)]$ with respect to $a_0, a_1, ..., a_{T-1}$. The idea of the constructed MDP setting is to find a strategy to maximize the probability of exiting the maze.

The action space includes all the possible actions that the agent can perform. In this case it is to stay still or move left, right, up or down. Thus, $\mathbb{A} = \{\text{stay}, \text{left}, \text{right}, \text{up}, \text{down}\}$

The state space includes all the possible pair positions that the agent and minotaur can be in besides the cases where they are on the same position (in which case the agent is eaten) or the agent is in the exit position (in which case the agent has exited). To compensate for those cases, the state space includes two states for when the agent is in exit or the agent has been eaten by the minotaur. Thus, $\mathbb{S} = \{(i, j, k, l) : i, k \in \mathbb{N}[0, 6], j, l \in \mathbb{N}[0, 7] \text{ and } (i, j) \text{ not in wall and } (i, j) \text{ not in exit and } (i, j) \neq (k, l)\} \cup \{\text{Eaten}\} \cup \{\text{Exit}\}$.

Now to compute the transition probabilities. First, as clarification; if an action is performed by the agent that would try to move the agent out of bounds or in to the wall then the agent will remain in the same position. If we are in the eaten or exit state, then we will remain in those states respectively, regardless of which action we perform and cannot transition in to other states. Thus,

- $p(s_{t+1}|s_t = \text{Eaten}, a_t) = \begin{cases} 1, & \text{if } s_{t+1} = \text{Eaten} \\ 0 & \text{otherwise} \end{cases} \quad \forall a_t \in \mathbb{A}$
- $p(s_{t+1}|s_t = \text{Exit}, a_t) = \begin{cases} 1, & \text{if } s_{t+1} = \text{Exit} \\ 0 & \text{otherwise} \end{cases} \quad \forall a_t \in \mathbb{A}$

If we are in a state $s_t = (i, j, k, l)$, assuming that we do not reach an Exit or Eaten state, then the actions of the agent determines the new position of the agent $(i, j)'$ and the random walk result of the minotaur determines the new position of the minotaur $(k, l)'$. If we denote N as the number of new positions that the random walk can lead the minotaur to from the current sate s_t, then if the agent performs action a_t, there will be $N \in \{2, 3, 4\}$ possible new states that we can transition into which corresponds to the new position of the agent (from performing a_t) and the new positions of the random walk, each with probability $1/N$. Thus,

- $p(s_{t+1} = (i, j, k, l)'|s_t = (i, j, k, l), a_t) =$
  $\begin{cases} 1/\text{N}, & \text{if } (i, j)' \text{ is reachable from } (i, j) \text{ when performing } a_t \text{ and } (k, l)' \text{ is reachable for the minotaur.} \\ 0 & \text{otherwise} \end{cases}$
  $\forall a_t \in \mathbb{A}$

Following upon previous paragraph but now tackling the case for which the new state s_{t+1} is Eaten. Similar holds for this case too. If the agent performs an action a_t then the only way we reach an Eaten state is if the minotaur's random walk results in that their positions coincides (which happens with probability $1/N$ given if this state is within reach or a probability of 0 else). Thus,

- $p(s_{t+1} = \text{Eaten} | s_t = (i,j,k,l), a_t) =$
  $\begin{cases} 1/\text{N}, & \text{if Eaten is reachable from } (i,j,k,l) \text{ for some outcome of the random walk when performing } a_t \\ 0 & \text{otherwise} \end{cases}$
  $\forall a_t \in \mathbb{A}$

Finally, we have the case for which we are in state $s_t = (i,j,k,l)$ and we reach the Exit state. Now, given that we are in a state such that an action a_t would transition us into the Exit state, this will not take into account the movement of the minotaur (thus reaching the exit state with probability of 1) except for when the minotaur also moves to the exit position. The minotaur will also move to the exit position given that it is reachable with probability $1/N$. Thus,

- $p(s_{t+1} = \text{Exit} | s_t = (i,j,k,l), a_t =$
  $\begin{cases} 1, & \text{if Exit is reachable from } s_t \text{ when performing action } a_t \text{ and exit position is not in reach for minotaur} \\ \text{1-1/N}, & \text{if Exit is reachable from } s_t \text{ when performing action } a_t \text{ and exit position is in reach for minotaur} \\ 0, & \text{otherwise} \end{cases}$
  $\forall a_t \in \mathbb{A}$

Now for the rewards. If the agent is in a non-eaten and non-exit state s_t at time t and performs an action a_t such that the next state s_{t+1} is the exit state, then they are rewarded by 0.

- $r(s_t, a_t) = 1$, $\forall s_t = (i,j,k,l), \forall a_t \in \mathbb{A}$, such that $s_{t+1} = \text{Exit}$

If the agent is in a non-eaten and non-exit state s_t at time t and performs an action a_t such that the next state s_{t+1} is the eaten state, then they are rewarded by -1000.

- $r(s_t, a_t) = 0$, $\forall s_t = (i,j,k,l), \forall a_t \in \mathbb{A}$, such that $s_{t+1} = \text{Eaten}$

If the agent is in a non-eaten and non-exit state s_t at time t and performs an action $a_t \neq$ stay such that the next state s_{t+1} is neither an exit or eaten state and the agent is at a new position compared to previous state, then they are rewarded by -1.

- $r(s_t, a_t) = 0$, $\forall s_t = (i,j,k,l), \forall a_t \in \mathbb{A} \backslash \{\text{stay}\}$, such that $s_{t+1} = (i,j,k,l)'$ and $(i,j)' \neq (i,j)$

If the agent is in a non-eaten and non-exit state s_t at time t and performs an action $a_t \neq$ stay such that the next state s_{t+1} is neither an exit or eaten state and is at the same position as previous state, then they are rewarded by -100.

- $r(s_t, a_t) = 0$, $\forall s_t = (i,j,k,l), \forall a_t \in \mathbb{A} \backslash \{\text{stay}\}$, such that $s_{t+1} = (i,j,k,l)'$ and $(i,j)' = (i,j)$

If the agent is in a non-eaten and non-exit state s_t at time t and performs the action $a_t =$ stay such that the next state s_{t+1} is neither an exit or eaten state and is at the same position as previous state, then they are rewarded by -1.

- $r(s_t, a_t) = 0$, $\forall s_t = (i,j,k,l)$, for $a_t = \text{stay}$, such that $s_{t+1} = (i,j,k,l)'$ and $(i,j)' = (i,j)$

Finally, if the agent is in a eaten or exit state, then regardless of action they will receive a reward of -1.[1] Thus,

- $r(s_t, a_t) = 0$, for $s_t = \text{Eaten or Exit}, \forall a_t \in \mathbb{A}$,

To summarize, we assign a positive 1 reward only if the agent enters the exit state successfully, all the other rewards equal 0

[1] At first we kept this value to 0, meaning that we should stay in the exit state for as long as possible. However, since the problem description wants us to find the maximum probability of escaping the exit (not as soon as possible) we modified this to -1. Although the policy changes for these two cases (for -1 the agent is staying in start position for a while until finally moving to exit, while for 0 the agent is immediately going towards the exit), the total probability of exiting does not.

1.2 1b

The policy that maximizes the probability of leaving the maze can be found using dynamic programming since the problem has a finite horizon (T = 20). The policy is illustrated by simulating a game where the positions of the agent and minotaur are updated at each time step. This can be seen in the file "simulation_problem_b.gif" that can be found in the zip-file. In the simulation, we see that the agent chooses to stay in the start position for a few time steps[2] and then it moves by taking the shortest path towards the exit position.

1.3 1c

The probability of exiting the maze was found by computing the final state with the help of the transition probabilities, i.e.

$$x_T = Px_{T-1} = PPx_{T-2} = ... = P^T x_1 \tag{1}$$

for each value of the end horizon T. Here P denotes the transition probability matrix and x_t is the state vector at time t (the probability of being in each state at the time t). x_1 was initialized to have the probability of 1 in the state where the minotaur is in position B and the agent in position A and 0 elsewhere. The final probability was found by looking taking the value of x_T[Exit]. The plot for this probability can be seen in Figure 1. We see that the probability of exiting the maze is 0 up until T = 15 for which it is 1. The reason it is 0 before T = 15 is because the agent needs a minimum of 15 steps in order to reach the exit position. When T $\geq$ 15, the agent will reach the exit with probability 1. The reason for this can be divided into two cases. The first case is when T = 15, then the policy will tell the agent to immediately go to the exit and since the sum of the starting coordinates of the minotaur is odd and the sum of the starting coordinates of the agent is even - the two will never end up in the same position as long as the agent keeps moving (which it will do since it needs to move every step in order to make to the exit before the time horizon), and therefore it reaches the exit with probability 1. The other case is when T$\geq$16. In this case, the policy tells the agent to stay for T-15 steps if T-15 is even and T-15-1 step if T-15 is odd (these two cases arises due to that the agent should not start moving when the sum of the minotaur's coordinates is even). After that the same argument as the last case holds. Thus the probability of exiting is 1 for all T$\geq$15

[2]The agent decides to make an action $a_t \neq$ stay when the sum of the coordinates of the minotaur $k + l$ is odd since then the minotaur and agent will never be able to be in same position as long as the agent keeps moving (since the sum of the agent's starting position is 0 which is even). This is because the minotaur always has to move in some direction. The policy also takes into account that the agent must be able to reach exit withing the time horizon.

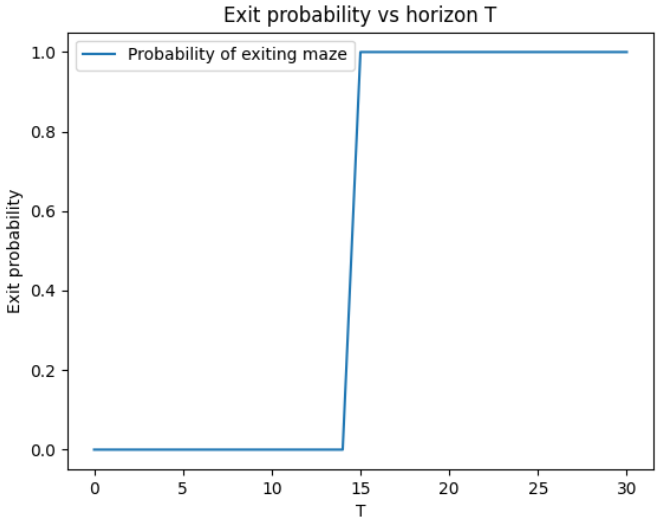

Figure 1: The probability of escaping the maze for different values of the end horizon

1.4 1d

According to the definition of geometric distribution and its property, denoting the probability of death due to poison as p, then the probability of life span T_l is given by

$$\mathbb{P}(T_l = t) = (1-p)^{t-1}p$$

From the second exercise session, we know that this problem can now be modelled as an infinite discounted horizon problem.
So the usual discount factor γ is in this case $\gamma = 1 - p$.
Thus, the objective is to maximize $\mathbb{E}[\sum_{t=1}^{\infty} \lambda^{t-1} r(s_t, a_t)]$ with respect to $a_0, a_1, ...$ and where $\lambda = 1 - p$. Also, we know that if $T \in Ge(p)$ then $\mathbb{E}[T] = 1/p = 30$, which gives us that $1 - p = \lambda = 29/30$.

The rewards, transition probabilities, action space and state space remains the same for the problem, only the objective function has changed.

We were considering adding a new state "dead" which would be the state that the agent goes to if they die from poison. With this, we would in each non-eaten, non-exit and non-dead state have to add a transition probability that the agent dies from poison which would be with probability $p = 1/30$ and that the agent survives (and therefore the transition probabilities from a) applies) with probability $1 - p = 29/30$ and if the agent is in a dead state it would remain there with probability 1. However since this probability applies for *all* states in which actions performed has an impact on the next state (states on the form (i, j, k, l)) and that the objective function has the discount in mind, meaning that later rewards are "more" discounted than earlier (meaning the best strategy is to reach the goal (which gives the highest reward) as soon as possible) - the optimal policy will still be found without this state and these transition probabilities being considered.

1.5 1e

First the policy was computed using value iteration. The policy was to reach the goal as soon as possible (no more staying in the start position like in b) and c)). Then 10000 games were simulated and the probability that the last state was the exit state was averaged over 10 runs. The result was: The agent has thus around a 62% chance of making it out of the maze alive.

One thing that is interesting to note is that since the sum of the starting coordinates of the minotaur $k + l$ is odd and the sum of the starting coordinates of the agent $(i + j)$ is even, as discussed in

μ	σ
0.6218	0.0053

Table 1: The mean (μ) and standard deviation σ of the probability to exit the maze over the 10000 simulated games computed over 10 runs.

previous questions, the agent and minotaur will never end up in the same position as long as the agent moves (since the minotaur has to move). Therefore, this probability can actually be computed analytically as it is just the probability that the agent survives until it makes it to the exit, that is,

$$\mathbb{P}(T \geq 16) = 1 - \mathbb{P}(T \leq 15) = 1 - \sum_{t=1}^{15} \mathbb{P}(T = t) = 1 - \mathbb{P}(T = 1) + \mathbb{P}(T = 2) + ... + \mathbb{P}(T = 15) \approx 61.5\% \quad (2)$$

Which is within one standard deviation of the mean that we got from our simulation!

1.6 1f

1) First, let's call the policy that we are trying to learn from the rewards received based on the actions chosen the target policy. Also, let's call the policy that the agent is using to choose its actions in a given state the behaviour policy. In an off-policy learning method, we try to learn the target policy from data that has been generated by a behaviour policy such that the behaviour policy and target policy are different. In an on-policy learning method, the target policy and behaviour policy are instead the same, thus we try to learn the behaviour policy.

2) The convergence conditions for the Q-learning algorithm is that the sum of the learning rates α_t must diverge but the square sum of α_t must converge: $\sum_t \alpha_t = \infty$, $\sum_t \alpha_t^2 < \infty$. Also, the behaviour policy π_b must visit each state action pair infinitely often. We also require that the discount factor is $0 < \lambda < 1$.
The convergence conditions for the SARSA algorithm is that the sum of the learning rates α_t must diverge but the square sum of α_t must converge: $\sum_t \alpha_t = \infty$, $\sum_t \alpha_t^2 < \infty$. Also, the policy π_t needs to be the ϵ greedy policy with respect to the learnt Q-function. We also require that the discount factor is $0 < \lambda < 1$.

1.7 1g

The problem is just like in d) a discounted infinite horizon problem. Using the same derivation as in d) for computing λ we have now get $\lambda = 1 - p = 1 - 1/50 = 49/50$. Now, the other new things that has to be taken into consideration is that we need to pick up the keys before reaching the exit, as well as the movement of the minotaur has changed.

The action space as well as the rewards remain the same as in d).

The state space is the same as in d) and a) but has now for all the states of the form (i, j, k, l) and extra parameter added (i, j, k, l, m) where $m = \begin{cases} 1, & \text{if the agent has the keys} \\ 0 & \text{otherwise} \end{cases}$.
Reaching the position C immediately sets $m = 1$ (thus we can not be in position C without the keys, i.e. $((i_c, j_c, k, l, 0) \notin \mathbb{S}$ where i_c, j_c are the coordinates of where the keys are in the maze).

The transition probabilities has to change quite a bit due to the minotaurs new movement as well as the new states. Now, if are in a state $s_t = (i, j, k, l, m)$, and perform an action a_t such that the new position of the agent is $(i, j)'$ and the position C is not within reach when performing this action then there will be a 35% chance that the minotaur moves towards the agent and 65% that it moves randomly. If we let once again N denote the number of available actions of the minotaur and we let M denote the number of moves the minotaur could do that would be conisdered to be walking towards the agent (it could be 1 or 2 moves, 2 moves is when the agent is in the diagonal of the minotaur). Thus,

- $p(s_{t+1} = (i,j,k,l,m)' | s_t = (i,j,k,l,m), a_t) = \begin{cases} 0.65/(\text{M}), & \text{if CaseA} \\ 0.35/(\text{N}), & \text{if CaseB} \quad \forall a_t \in \mathbb{A} \\ 0 & \text{otherwise} \end{cases}$

where CaseA $= (i,j)'$ is reachable from (i,j) when performing a_t and $(k,l)'$ is reachable for the minotaur $(k,l)'$ is the position reachable for the minotaur that is the closest to (i,j) and CaseB $= (i,j)'$ is reachable from (i,j) when performing a_t and $(k,l)'$ is reachable for the minotaur $(k,l)'$ is the position reachable for the minotaur that is not the closest to (i,j) .

The rewards has changed only in the way that we need to consider the new m parameter. The only difference is that we exit the maze by having the keys when we reach position B which means that m must be 1 in order to get the reward for exiting:

- $r(s_t, a_t) = 0$, $\forall s_t = (i,j,k,l,m), \forall a_t \in \mathbb{A}$, such that $s_{t+1} = \text{Exit}$, and $m = 1$

Other than that, all the rewards mentioned in a) hold for all allowed values of m.

1.8 1h

1.8.1 1)

Pseudocode for the Q-learning algorithm:
initialize Q
$Q[s,a] = r[s,a], \forall s, a$
initialize $n(s,a)$, the number of visits to (s,a)
$n[s,a] = 0 \forall s, a$
# begin episodes
for episode in episodes:

    s_{t+1} = get_start_state
    while s_{t+1} is not terminal state:
        #update new state
        $s_t = s_{t+1}$
        if agent dies from poison:
            break
        # get the action according to epsilon greedy policy
        a_t = epsilon_greedy(s_t)
        #update n
        $n[s_t, a_t] + +$
        # get the reward and future state
        $reward = r[s_t, a_t]$
        s_{t+1} =compute_new_state(s_t, a_t)
        # update Q value.
        $Q[s_t, a_t] = Q[s_t, a_t] + \alpha_t(n)(reward + \gamma max(Q[s_{t+1}]) - Q[s_t, a_t])$
$V = max(Q)$
policy $= argmax(Q)$
return V, policy

For clarification: here, α is the learning rate, $r[s,a]$ is the instantaneous reward the agent receives by getting to state s and performing action a, max and $argmax$ is the max and $argmax$ over all the actions, γ is the discount factor

1.8.2 2)

We see that when choosing epsilon small, e.g. 0.1, then the convergence speed is very low. By changing the value to 0.2, the speed is enhanced. Figure 2 and Figure 3 compares the different epsilon values. The problem may lie in the fact that small epsilon will not explore enough especially with a not good initialization of the q value function..

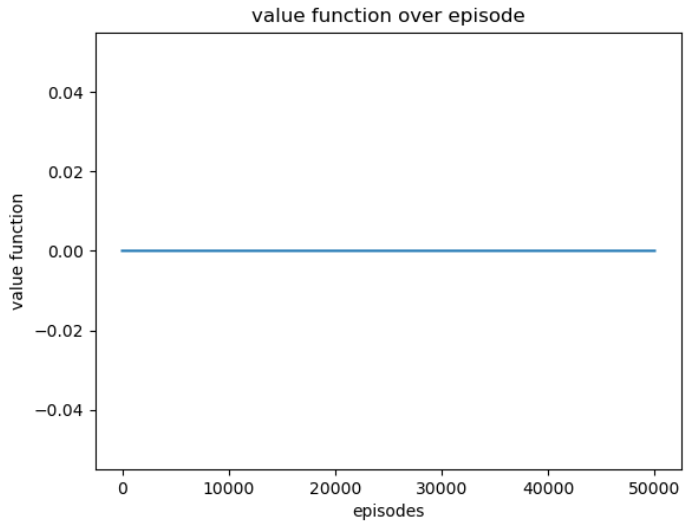

Figure 2: value function of start position with Q function $\epsilon : 0.1$

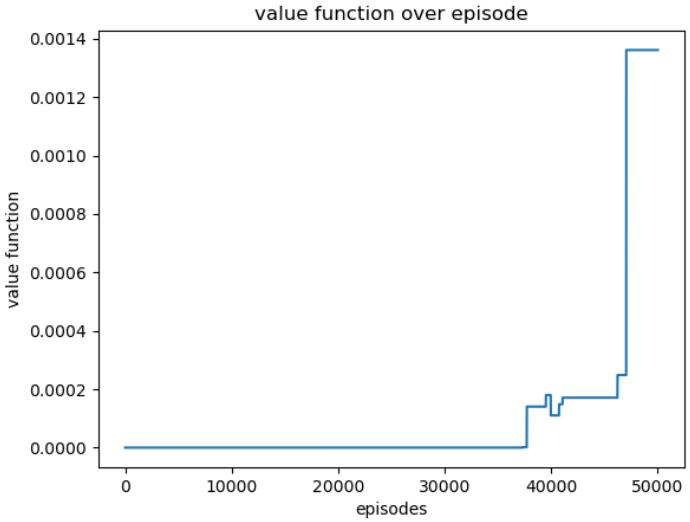

Figure 3: value function of start position with Q function $\epsilon : 0.2$

The visualisation of a simulation can be found under the name *qLearn_simulate.gif*. We use an additional color purple to indicate the states that the agent has already collected the key. The simulation will show that the agent will first go to the key position and pick the key and go to the exit.

A good guessed Q-function at initialisation will definitely improve the convergence speed. We have provided an example use the value calculated by a previous calculation with visit of states count accumulating (cf. Figure 4), so the initialized Q-function is better than the initialisation as the reward function matrix. The result is shown in Figure 5. We could see that the value starts at around -11 and exposes less variance towards the final value.

1.8.3 3)

When we fix the ϵ and vary the α, different convergence speeds could be seen.

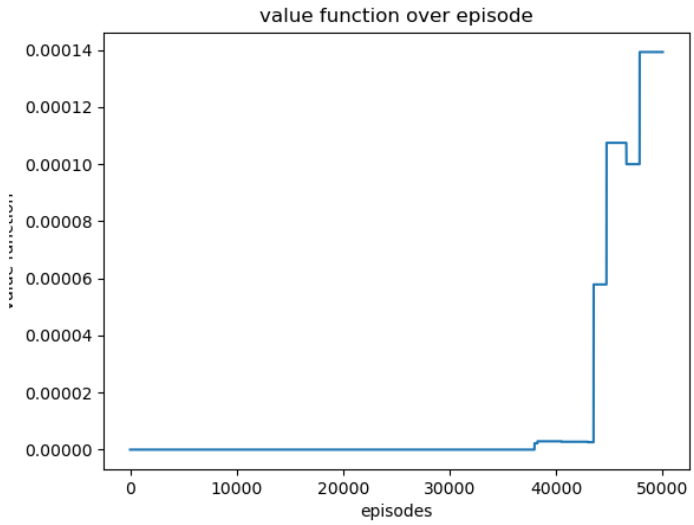

Figure 4: value function of start position with Q function initialized from previous result

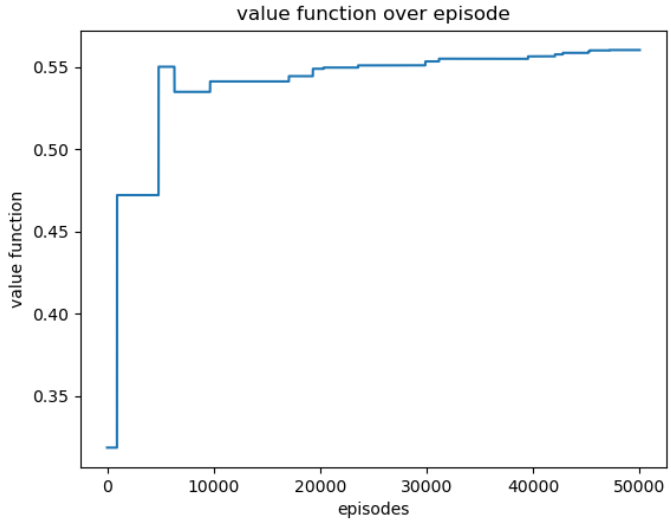

Figure 5: value function of start position with Q function initialized from previous result

Varying the alpha will also influence the convergence speed. We see that at higher alpha, so the step size decreases more rapidly, the convergence speed is lower, because the learning rate is penalized at long run, as supposed.

1.9 1i

1.9.1 1)

Pseudocode for the SARSA algorithm:
initialize Q
$Q[s, a] = r[s, a], \forall s, a$
initialize $n(s, a)$, the number of visits to (s, a)
$n[s, a] = 0 \forall s, a$
# begin episodes
for episode in episodes:

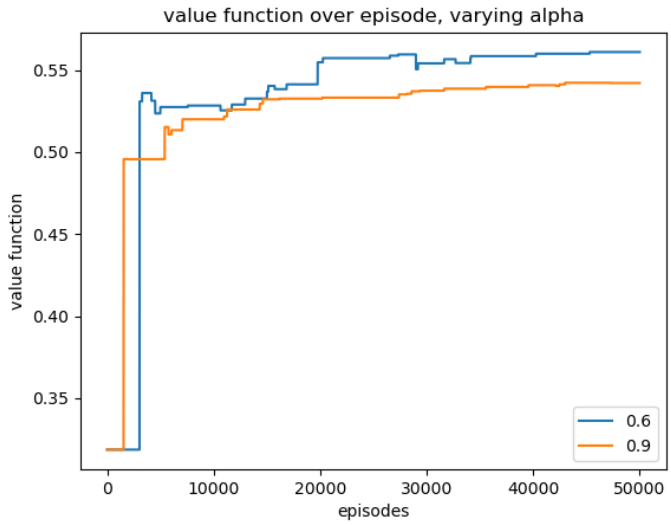

Figure 6: value function of start position with varying α

initialize state and compute action in start state
s_{t+1} = get_start_state
a_{t+1} = epsilon_greedy(s_{t+1})

while s_{t+1} is not terminal state:
#update new state and update action
$s_t = s_{t+1}$
$a_t = a_{t+1}$
if agent dies from poison:
break
#update n
$n[s_t, a_t] + +$
get the reward, future state
$reward = r[s_t, a_t]$
s_{t+1} =compute_new_state(s_t, a_t)
now also get the action in the new state according to epsilon greedy policy
a_{t+1} = epsilon_greedy(s_{t+1})
update Q value.
$Q[s_t, a_t] = Q[s_t, a_t] + \alpha_t(n)(reward + \gamma Q[s_{t+1}, a_{t+1}] - Q[s_t, a_t])$
$V = max(Q)$
policy = $argmax(Q)$
return V, policy

For clarification: here, α is the learning rate, $r[s, a]$ is the instantaneous reward the agent receives by getting to state s and performing action a, max and $argmax$ is the max and $argmax$ over all the actions, γ is the discount factor.

1.9.2 2)

Again, we see that the epsilon affects the convergence speed.

For $\epsilon = 0.2$, Figure 7 shows the result with q value initialized to 0 and Figure 8 uses the previous simulated result Figure 9 shows third stage simulated. We see the

The simulated result can be found in the name *sarsa_1stage_sim.gif*

For $\epsilon = 0.1$. We see that the convergence speed is much slower. With Figure 10, Figure 11 and Figure 12 generated similarly.

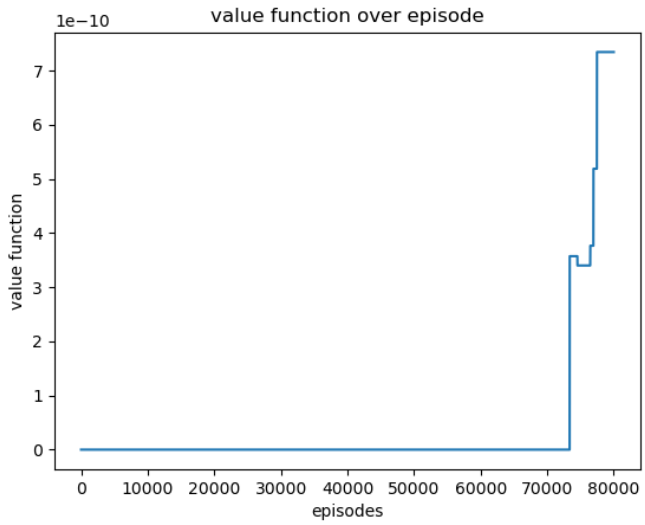

Figure 7: value function, $\epsilon = 0.2$, from scratch initilized

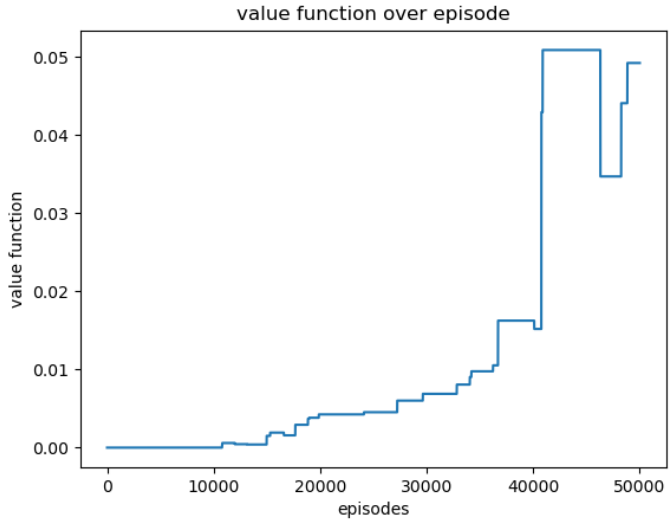

Figure 8: value function, $\epsilon = 0.2$, second stage

1.9.3 3)

With varying deltas, we have plotted the result with Q value initialized with the value calculated from a previous experiment. The plot is found in Figure**??**. We see that with different deltas, all converge to some stable values, however the delta chosen at 0.9 seems to perform the best. In this case, when delta is larger than the α the performance is better. Perhaps we should penalize more the epsilon w.r.t the learning rate.

1.10 1j

The average probability (over 100 runs) of leaving the maze calculated using Q learning and SARSA can be seen from the previous two sections. The value is around 0.56. We also prove the result using the value iteration and the result match with each other. This should correspond to the Q-value of the initial state because the non-negative reward is the only value that contributes to the initial state value function, which is the expected reward, which is the exit probability. 2.

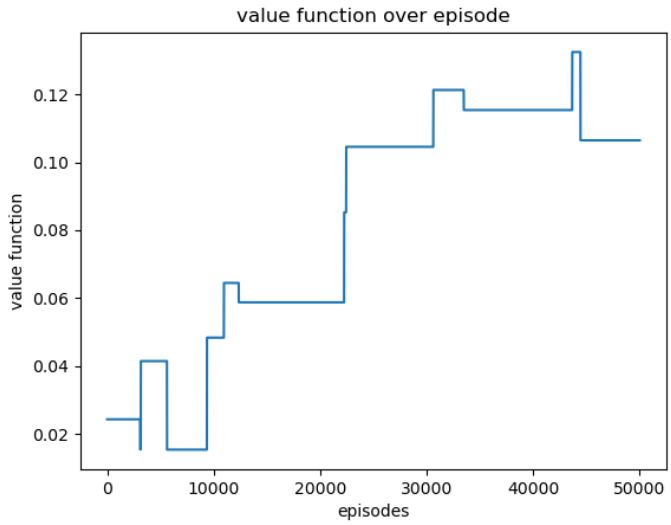

Figure 9: value function, $\epsilon = 0.2$, third stage

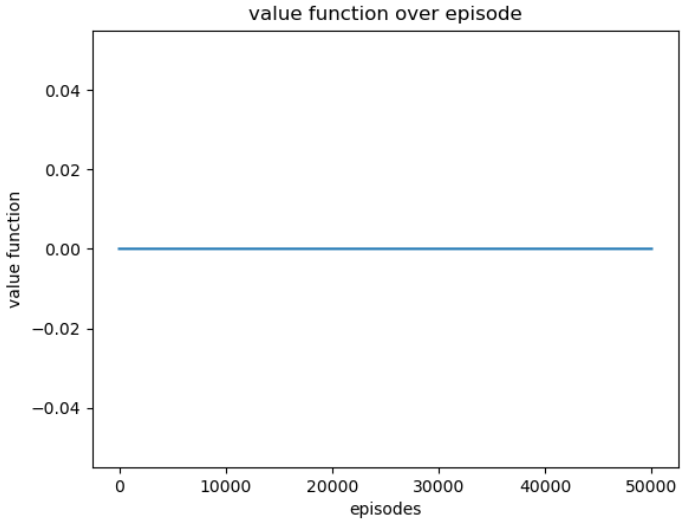

Figure 10: value function, $\epsilon = 0.1$ first stage

The average probability (over 100 runs) of leaving the maze using a policy computed through SARSA can be seen in Table 3.

Discussion about if the probabilities are close to the Q-value of the initial state if so/not explain why: To be added..

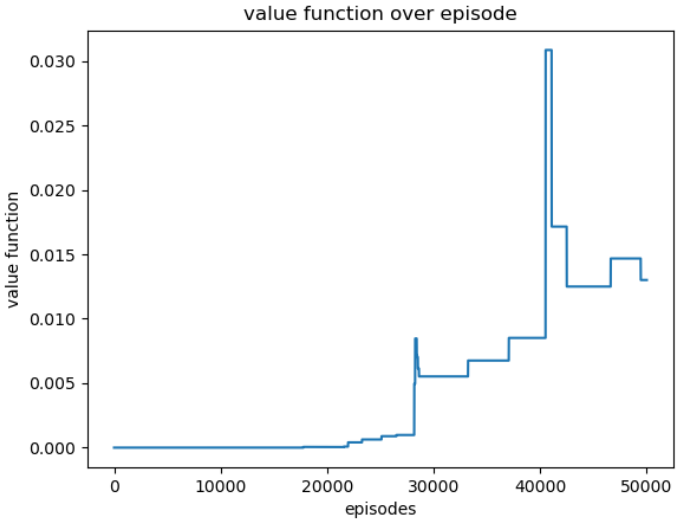

Figure 11: value function, $\epsilon = 0.1$, second stage

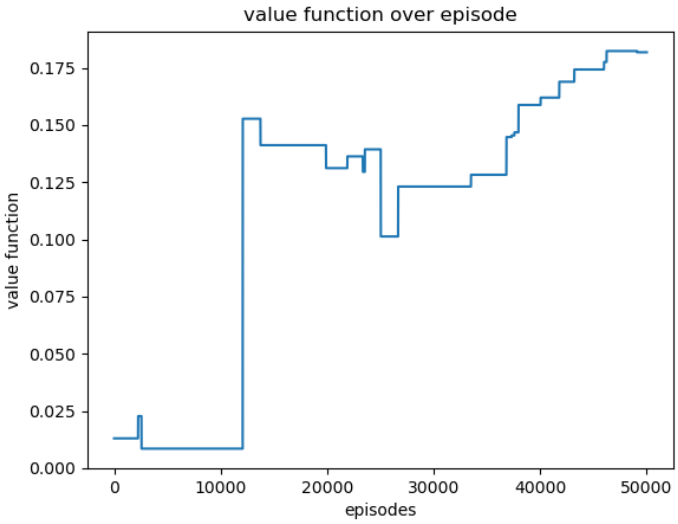

Figure 12: value function, $\epsilon = 0.1$, third stage

μ	σ
0.568	0.005

Table 2: Estimated probability of exiting the maze using a policy computed through Q-learning. μ is the mean and σ is the standard deviation.

μ	σ
0.568	0.006

Table 3: Estimated probability of exiting the maze using a policy computed through SARSA. μ is the mean and σ is the standard deviation.

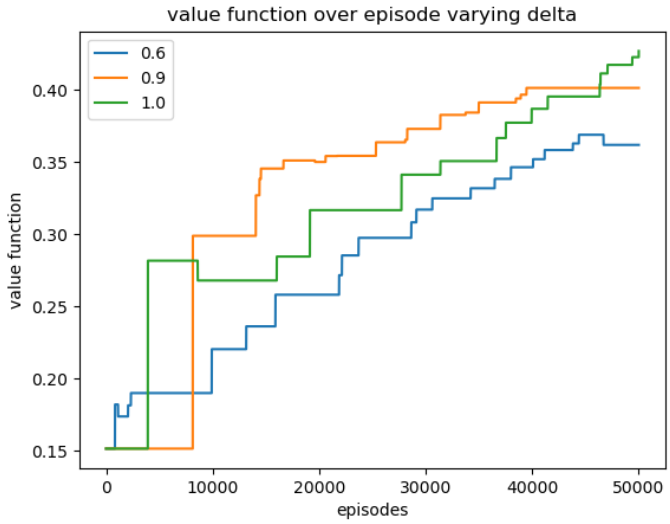

Figure 13: value function, varying deltas

2 Problem 2

2.1 c)

We use the following parameters

$episodes = 300, \alpha = 0.001, \gamma = 0.99, \lambda = 0.8, \delta = 0.9$ where the δ controls the decreasing ϵ in the learning, as in problem 1. For the basis, we have chosen the vectors (columnwise)

$$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

We have used the SGD modification with momentum, clipping the eligibility trace.

2.2 d)

To be added..

2.2.1 1)

Figure 14 shows the episodic rewards. We see that the average increases and reach near the value around -120

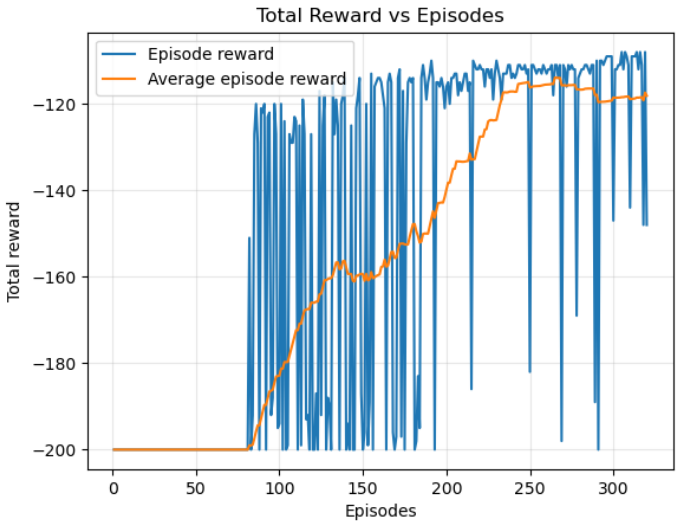

Figure 14: Rewards over episodes

2.2.2 2)

Figure 15 shows the value function plotted. We have normalized the s_1 and s_2 to the range $[0, 1]$.

The plot shows that the highest value function appear at the spot where s_1 and s_2 are positive, which corresponds to the situation that the car is near the pole and the velocity is to the right, which makes sense.

2.2.3 3)

Figure 16 shows the policy plotted.

The policy plotted shows that the policy is more dependent on the velocity which makes sense because with initial velocity, the car will fully utilize that initial momentum so we see that the policy usually follows the same direction as the velocity direction points towards.

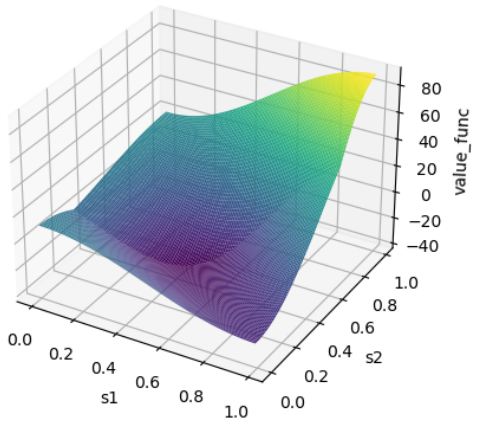

Figure 15: Rewards over episodes

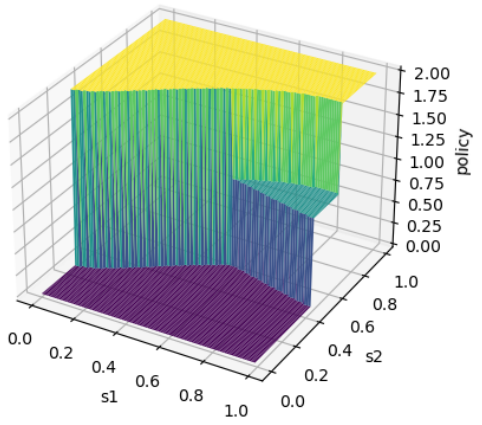

Figure 16: Rewards over episodes

2.2.4 4)

We have tried both with zero vectors added and not. The result seems similar. The intuitive explain is that adding a zero vector will only add an offset for all the Q value function evaluated, so it does not affect much the optimal policy.

2.2.5 5)

Figure 17 shows the value function plotted.

2.3 e)

The rewards with different alphas is shown in Figure 18. We see that as the learning rate grows large, the average tends to be lower, it seems that the learning rate is too large. For different lambdas, the rewards are plotted in Figure 19. We see that lambda has some optimal values in between.

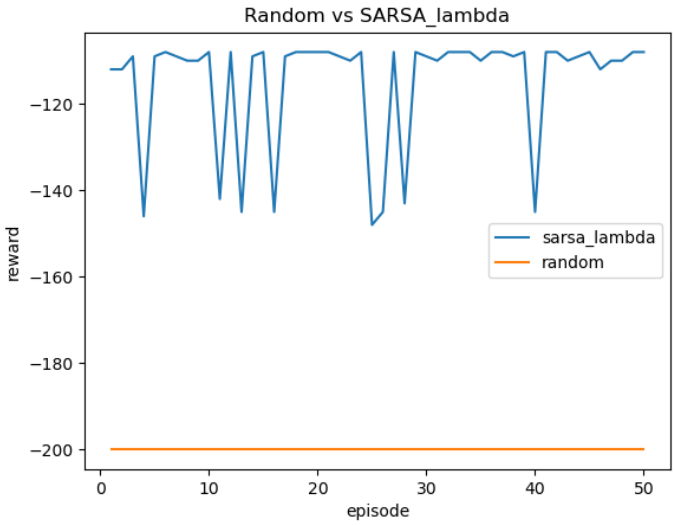

Figure 17: Random agent vs. SARSA lambda

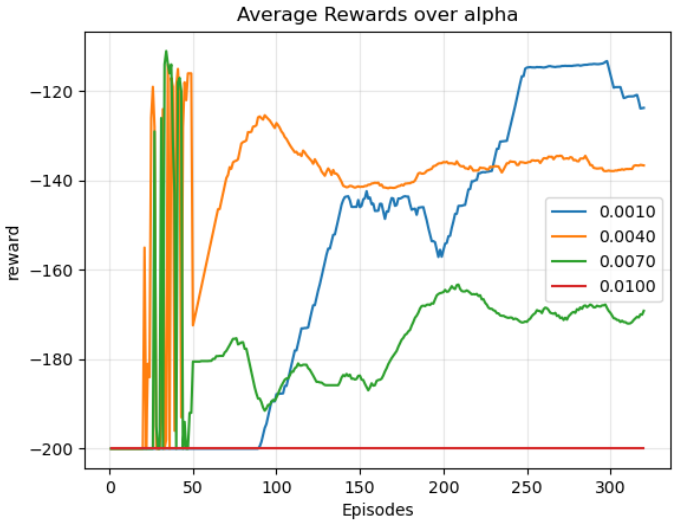

Figure 18: Rewards with different alpha

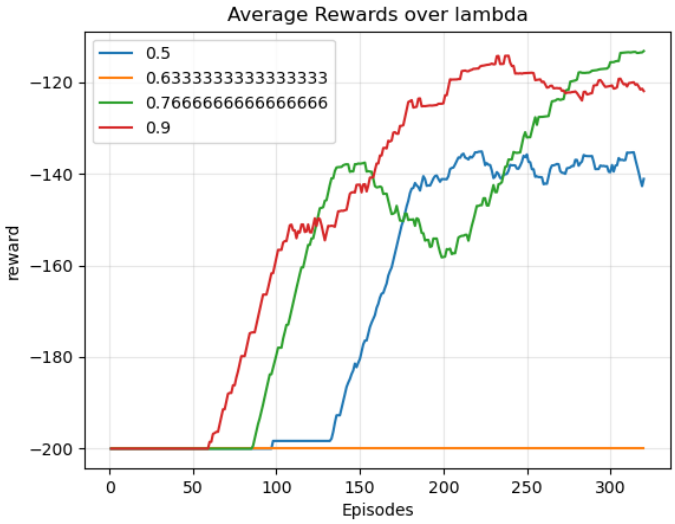

Figure 19: Rewards with different lambda