EL2805 — Reinforcement Learning

Name: João Almeida & Victor Sanchez
Student Number: 19990501-T210 & 19980429-T517
Due Date: November 28 2021, 23:59
Assignment: 1

Problem 1 - The Maze and the Random Minotaur

Mandatory Questions (a) - Basic Maze

Question (a) Problem formulation as an MDP

A MDP consists is composed by a State Space (S), an Action Space (A), Transition probabilities (T) and a Reward (R) $\tilde{M}$(S,A,T,R).

State Space:

$$\{1, ..., N_1\} \cup \{1, ..., N_2\} \cup \{Exit\} \cup \{GameOver\} = \{S1\} \cup \{S2\} \cup \{E\} \cup \{GO\} \quad (1)$$

$$Exit \in S_1 \quad (2)$$

$$S_3 = \{S_1\} \cup \{S2\} \quad (3)$$

N_1 : Number of free places in the maze; N_2 : Number of walls places in the maze;
S_1 : Path of the maze and admissible region where the player can move;
S_2 : Walls of the maze; S_3 : Admissible where the minotaur; quad GO : End of the game

Action Space:

$$\{left, right, up, down\} \cup \{stay\} = \{A_1\} \cup \{stay\} \quad (4)$$

$$A = \{A_1\} \cup \{stay\} \quad (5)$$

A : Action space of the player;
A_1 : Action space of the minotaur

Transition:

$$\begin{cases} p(s'|s,a) = 1, & s' \in S_1 \\ p(s'|s,a) = 0, & s' \notin S_1 \end{cases}, \quad s \in S_1, a \in A \quad (6)$$

$$p(s|s,a) = 1 - p(s'|s,a), \quad a \notin \{stay\}, s \in S_1 \quad (7)$$

$$\begin{cases} p(GO|s,a) = 1, & (s^*|s_1,a_1)_t \wedge (s^*|s_2,a_2)_t \\ p(GO|s,a) = 0, & otherwise \end{cases}, \quad s_1, s^* \in S_1, a_1 \in A, s_2 \in S_3, a_2 \in A_1, t > 0 \quad (8)$$

$$\begin{cases} p(s|s,a) = 1 \\ p(s'|s,a) = 0 \end{cases}, \quad a \in A, s \in \{GO\} \cap \{Exit\}, s' \neq s \quad (9)$$

Rewards:

$$r(Exit, stay) = 0 \tag{10}$$

$$r(GO, a) = -\infty, \quad a \in A \tag{11}$$

$$\begin{cases} r(s,a) = -1, & (s', s, a)s' \in S_1 \\ r(s,a) = -2, & (s', s, a)s' \notin S_1 \end{cases}, s \in S_1, a \in A \tag{12}$$

Mandatory Questions [(b)-(c)] - Dynamic Programming

Question (b) Problem formulation as an MDP

The implemented policy consists on computing the l_1 between the actual position of the minotaur and the future position of the player (intended move). If the result is 1, it means that with 25% probability the player can died to the minotaur in the next round. All the other actions are chosen randomly according to the maximum average reward value. As we can see in the figure 5 the first action (fig. 1), if the player want to go further, the probability of be in the same state as the minotaur is 0, so it chooses the action right. Although in the second action (fig. 2), both agents are head to head, so the player knows that the minotaur as to move in the next round. Note that the player could also choose to go up or left, but never stay. The third action (fig. 3) is basically the same of the previous action since the minotaur chose the same action of the player. Finally in the last figure (fig. 4) is possible to see that the player insisted again in going to the minotaur's previous position and in this case it worked as expected since the minotaur chose the action up, leaving the path free for the player to move with no constrains in the short future.

Question (c) Probability of leaving the maze with horizon = 30. Was there a difference if the minotaur is allowed to stand still?

With horizon=30 and the minotaur disallowed standing still, it was run the dynamic algorithm multiple times in sets of 1000 times per run. The result of the lowest run was 99,1%. On the other hand, if the minotaur is allowed to stand still the probability decreases until 91,7%, and this can be explain with help of the figure 5. For example in the second action (fig. 2) the player cannot choose the same action as before since with 20% probability the minotaur can in that state, so the safe move is either choose left or up. In the next instant the minotaur chose right, blocking for the player towards the goal. In addition, the probability of the minotaur moving further away from the maze is lower, which in a certain scale also "protects" more the exit.

First we can observe on (fig. 6) that if the horizon is inferior to 15, the player can not exit the maze, which is logical because the player needs at least 15 steps to reach the exit. After that threshold, the player can exit the maze not matter what the minotaur does.
Secondly, we observe that when the Minotaur can stay still, the player keeps moving to the exit to reach it. Which is logical because the threat is reduced.
Nevertheless, when the minautor can not stay sill there is a difference of parity : The number of steps the Minotaur has to do so as to reach the Player's shortest path is different to the number of steps the Player has to make. Thus, while the Player does not "stay" an odd number of times it is going to be for him. In fact, it takes the Player an odd number of steps to go to the exit and it takes the Minotaur an even number of steps to go to the exit too.

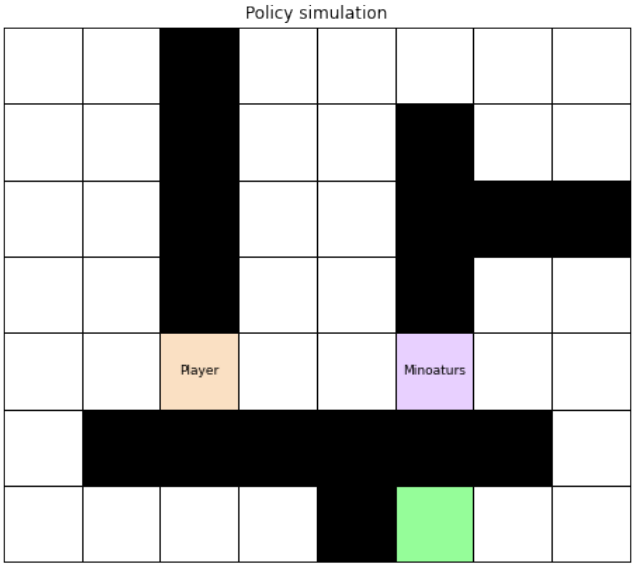

Figure 1: 1^{st} instant

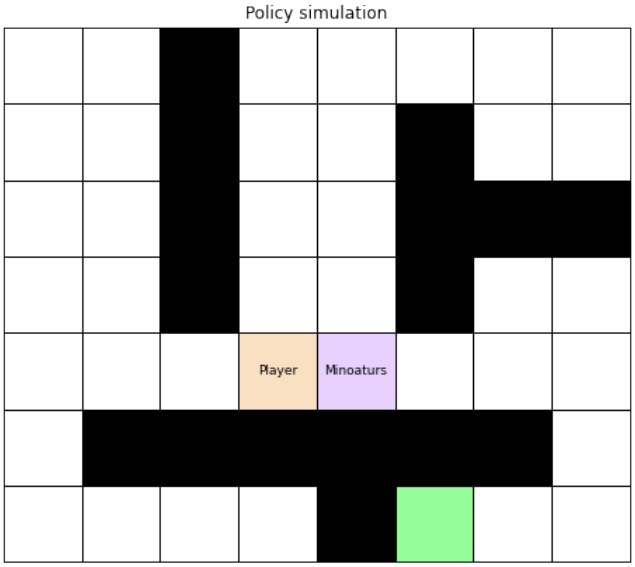

Figure 2: 2^{nd} instant

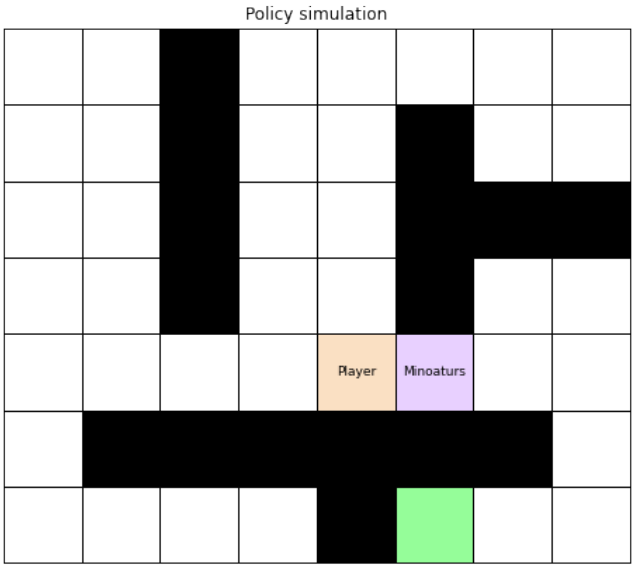

Figure 3: 3^{rd} instant

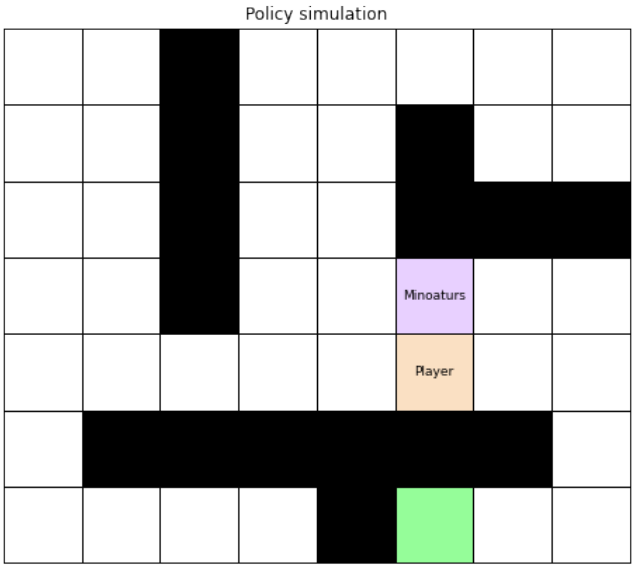

Figure 4: 4^{th} instant

Figure 5: Sequence of actions illustrating the policy

Modification given :
- We took into account your feedback and we implemented the algorithm for each time horizon

Mandatory Questions [(d)-(e)] - Value Iteration

Question (d) Problem modification to Value Iteration with life distributed with mean 30.

In this case, the value iteration algorithm is running until the difference between two consequent average reward matrices (B) is lower than a certain rate or the number of iterations is higher than 200.

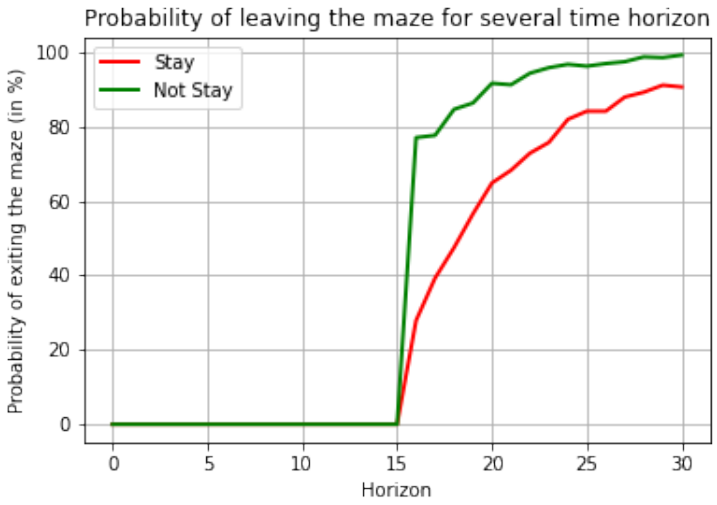

Figure 6: Probability of exiting the maze for the two behavior of the minotaur

Regarding the poison effect, an infite-time horizon MDP with discount factor λ can seen as a finite-time horizon geometrically distributed with probability p $= 1 - \lambda$. But we know that the mean of a such geometric law is $\frac{1}{p}$ which is giving p $= \frac{1}{30}$ and thus $\lambda = \frac{29}{30}$.
Moreover we take an ϵ factor equal to $1e - 4$ to have a low tolerance and thus, do more iteration to reach the exit.

Modification given :
- We deleted the simulation part on the value iteration algorithm
- We simplified the update of the V and the policy like in lab0
- We reduced epsilon so that the algorithm can provide a solution without having a threshold that is too important

Question (e) Probability of getting out alive using this policy by simulating 10000.

To realise so we implement a variable GEOM that follow a geometric distribution with the same parameter as explained in the former question. We then check for that law, with the nth if the player has exit or not. After 10000 runs, the player successfully exit in maze 7261 times (72.6%), which is actually quite a good result.

Modification given :
- We think that our mistake was on the way to compute the probability, we used an other with a special variable as explained below

Mandatory Questions [(f)-(g)] - Q-Learning and Sarsa

Question (f) Theoretical questions about Q-Learning and SARSA algorithms

Question (1) What does it mean that a learning method is on-policy or off-policy? In on-policy algorithms, the learning agent learns the value function according the current action derived from the policy currently being used. On the other hand, in off-policy algorithms, the

agent learns the value function according to the action derived from another policy and though independently of the agent's action.

Question (2) State the convergence conditions for Q-learning and SARSA.
It is guaranteed the converge of Q-Learning algorithm when the following conditions are meet:

- Each state-action pair must be visited infinitely often. In other words, each action must have a non-zero probability of being selected by the policy in every state ($\epsilon - greedy$ policy (where $\epsilon > 0$) ensures that this condition is satisfied);[1]
- In TD-learning and Q-learning, learning rate should decrease so that sum of α_k diverges and sum of α_k^2 converges.

The Q-Learning algorithm will converge to the q^* value which is leading to the optimal policy.

Regarding the SARSA algorithm, it will learn the optimal ϵ-greedy policy, i.e., the Q-value function will converge to a optimal Q-value function but in the space of ϵgreedy policy only (as long as each state action pair will be visited infinitely). We expect that SARSA will converge to the overall optimal policy.

Question (g) Modification the MDP reformulated in question (d)

For this question, we can add a fifth state on top with two elements:

$$S_5 = \{\text{No Key}, \text{Key}\} \tag{13}$$

Now, the reward matrix slightly change in the coordinate of the final state. With this being said, whenever we are in the state {No Key}, the final state is the position with maximum reward. Consequently, when the key is taken, and agent transits for the state {Key}, where the exit of the maze takes the final state and maximum reward state. Regarding the minotaur, the state and action space do not change but the transitions do. In this case the policy to choose the next action includes the position of the player, since now the minotaur can move towards the player with probability 35%.

This is simulated by dividing the problem in two different instances, firstly the MDP is formulated with eyes on the key state (like if this was the Exit state according to the problem formulation notation). Then the second instance with initial state the key state towards the exit with the life remaining after the end of the first instance, if any.

[1] This is true if we talk about convergence of Q-values, but to obtain the optimal policy the both GLIE conditions must be satisfy.

Bonus Questions [(h)-(j)] - Q-Learning and Sarsa

Question (h) Q-learning algorithm

As a mirror of what the group has done in question g), the problem of getting the key before leaving the maze was divided in two different instances, the first one between the start (0,0) state and the key state (0,7), and the second between the key state and the exit (5,6). Regarding MDPs with infinite horizon objective, the discount factor ensures that the that the unit reward decreases with time at geometric rate λ, therefore the discount factor used in the algorithms was 0.8. And the reward of reaching the goal state was increased to +10.

Question (1) Implementation in pseudo code

Algorithm 1 Q-Learning

Require: $Q(s,a) = r(s,a])$ ▷ Initialise Q(s,a) with the rewards
Ensure: $s \in S_1, a \in A$
  $y \leftarrow 1$
  $X \leftarrow x$
  $N \leftarrow n$
  while not Converge & episode < Max episodes **do** ▷ Repeat for each episode
    Initialise s
    while not final state & life > 0 **do**
      Take action a, observe R and s' ▷ R = r[s,a]
      Choose a from s using $\epsilon - greedy$
      $Q(s,a) \leftarrow Q(s,a) + alpha[R + \gamma max_{a^* \in A}(Q(s',a^*) - Q(s,a)]$
      s ← s'
    end while
  end while

Question (2) Influence of the ϵ parameter. Discussion on a proper initialisation of the Q-values may affect the convergence speed.

The problem B was solved for 2 different values $\epsilon = \{0.3, 0.5\}$. The convergence of the Q-values can be seen in the figures 7 and 8. In this figures, it is possible to see the convergence from the state A to state C in the image on the left and from state C to state B on the right

The convergence of the algorithm was defined when the euclidean norm of two consecutive Q-value matrices was less than 1e-2.

It was tested two different initialisation for the algorithm, not only setting the Q-matrix to 0 but also with the highest reward. It was observed that not only the rewards. According to the figure **??**, it is important to see the importance of the initialisation. However, the model must be tuned for each one. However for example when the initialisation was 0, we can see that the blue graph was the fastest one to converge ($\epsilon = 0.3$) when comparing to the +2000 iterations it needed when the Q matrix was initialised with the reward matrix. On the other hand, when it was used an optimistic initialisation (all values initialised with the highest reward - +10 of reaching the goal), we can see lets fluctuating of values for example in the second run, from the key state to the exit of the maze.

Note: since the tolerance is in absolute value, for higher values of the Q-matrix it might show more difficulties to stop, even though the relative difference is super low.

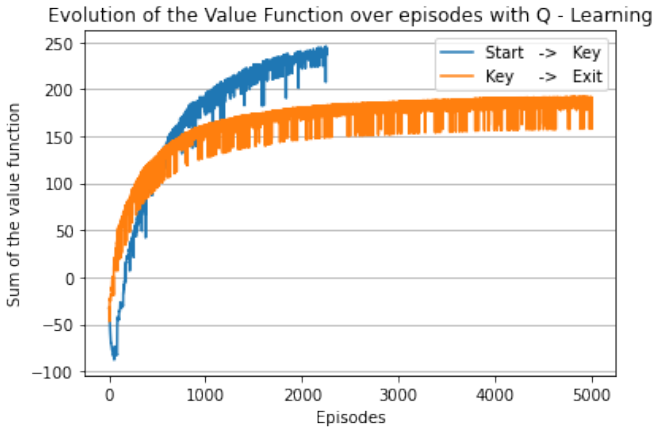

Figure 7: Convergence of Q-Values for $\epsilon = 0.3$ from state A to C and from state C to B

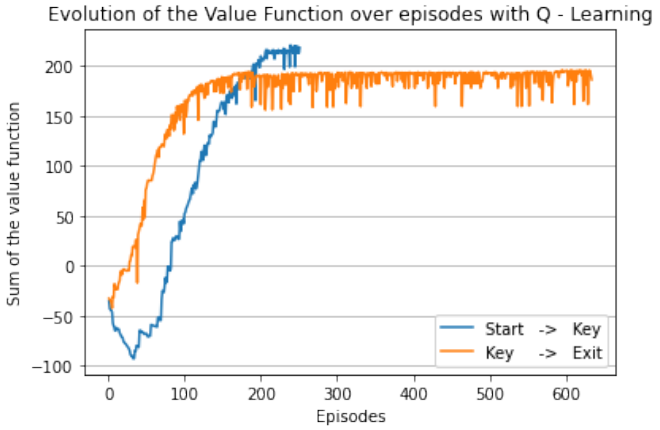

Figure 8: Convergence of Q-Values for $\epsilon = 0.5$ from state A to C and from state C to B

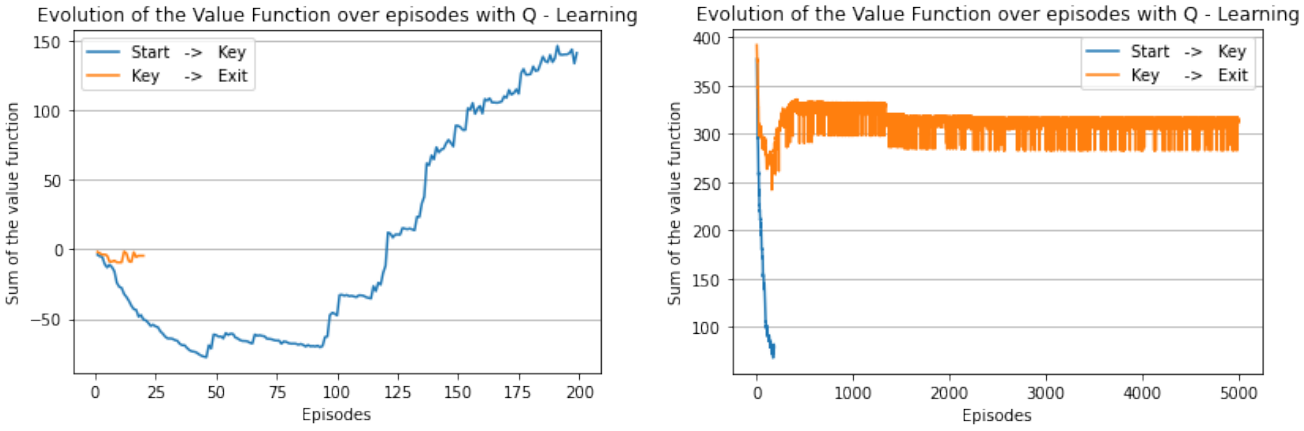

Figure 9: Importance of initialisation of the Q-matrix: on the left the initial Q-values are 0 and on the right the initial Q-values are 10

Question (3) Convergence of the algorithm

It was tested $\alpha = \{0.5, 0.8\}$. Since the denominator is proportional with $\alpha \in [0.5, 1]$ value, for values closer to one, the denominator will be higher and then the step size will be lower. It

is possible to confirm will help to the figures 10 and 11. Taking into consideration these 2 cases, when the α value was close to 0.5, the algorithm needed near 500 and 700 iterations to converge (fig. 10). On the other hand when α=1 (fig. 10), the algorithm, had not converge after 5000 episodes.

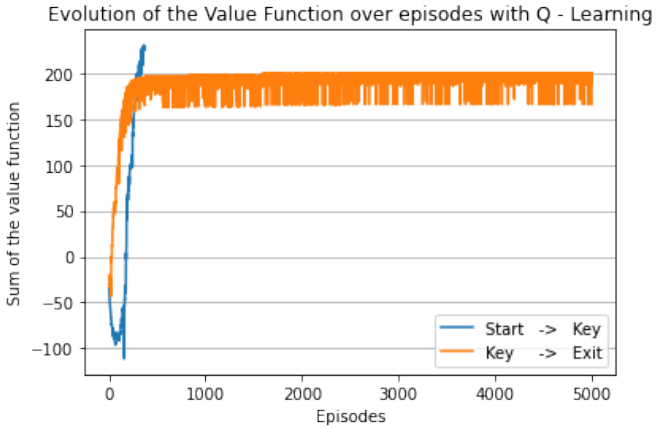

Figure 10: Convergence of Q-Values for $\alpha = 0.5$ from state A to C and from state C to B

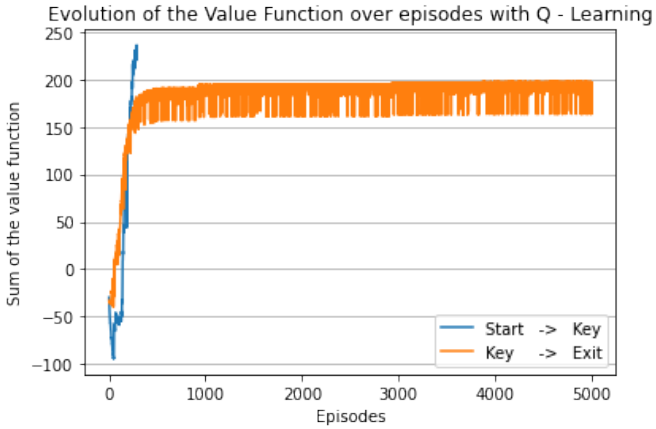

Figure 11: Convergence of Q-Values for α 1 from state A to C and from state C to B

Question (i) Implementation of SARSA Algorithm

For this algorithm the reward of reaching the goal was defined at 0!

Question (1) Implementation in pseudo code

Algorithm 2 SARSA

Require: $Q(s,a) = r(s,a])$ ▷ Initialise Q(s,a) to zero
Ensure: $s \in S_1, a \in A$
$prob \leftarrow randomvaluebetween0and1$
$As \leftarrow availableactions$
while not Converge & episode < Max episodes **do** ▷ Repeat for each episode
Initialise s
Take an action with $\epsilon - greedy$:
if $prob < epsilon$ **then** $a \leftarrow randomaction$
else$a \leftarrow argmax(Q(s,a))$
end if
Observe R and s' ▷ R = r[s,a]
Choose a' from s' using again $\epsilon - greedy$
$Q(s,a) \leftarrow Q(s,a) + alpha[R + \gamma max_{a \in A}(Q(s',a') - Q(s,a)]$
s ← s'
a ← a'
end while

Question (2) Influence of the ϵ parameter.

The problem B was solved for the value $\epsilon = \{0.2\ ,\ 0.1\}$. The convergence of the Q-values can be seen in the figures 12(for $\epsilon = 0.1$) and 13 (for $\epsilon = 0.2$) . In each figure, is possible to see the convergence from the state A to state C in the image on the left and from state C to state B on the right.

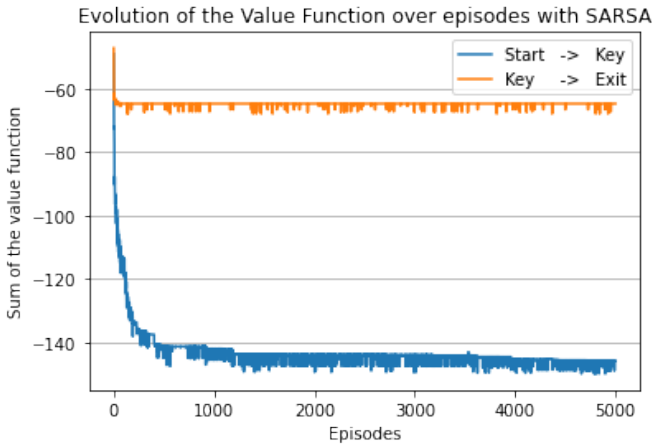

Figure 12: Convergence of Q-Values for $\epsilon = \{0.1\}$ from state A to C and from state C to B

Once again, it was tested the convergence speed for different initialisation of Q-values. Since the highest reward is 0, for this algorithm, it was test two variants of epsilon-greedy policy. According

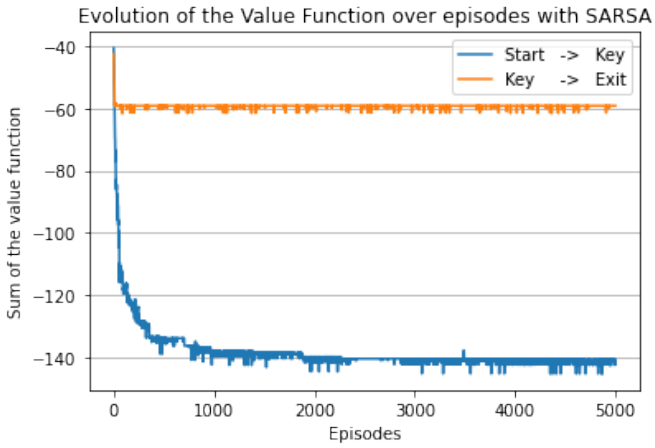

Figure 13: Convergence of Q-Values for $\epsilon = \{0.2\}$ from state A to C and from state C to B

to the figure 14, it is possible affirm that the algorithm has converge but for sure not for the optimal policy, however it was much smoother then the previous, since every state started without any advantages/disadvantages (according to the rewards). Note that the importance of the epsilon choice. In the left figure the random action is rarely chosen and it takes the two times the iterations when the random action is chosen often and therefore more states are visited resulting on a balanced Q-matrix. This can also be confirm with the instance from the key state to the exit, since when the greedy action is almost always chosen but the Q-value initialisation are 0, most of the states are not visit and so the algorithm stops quickly and the policy does not make sense, on the other hand, there are a few calculations that are a positive sign that the optimal policy could be reach (also if some other parameters were tuned).

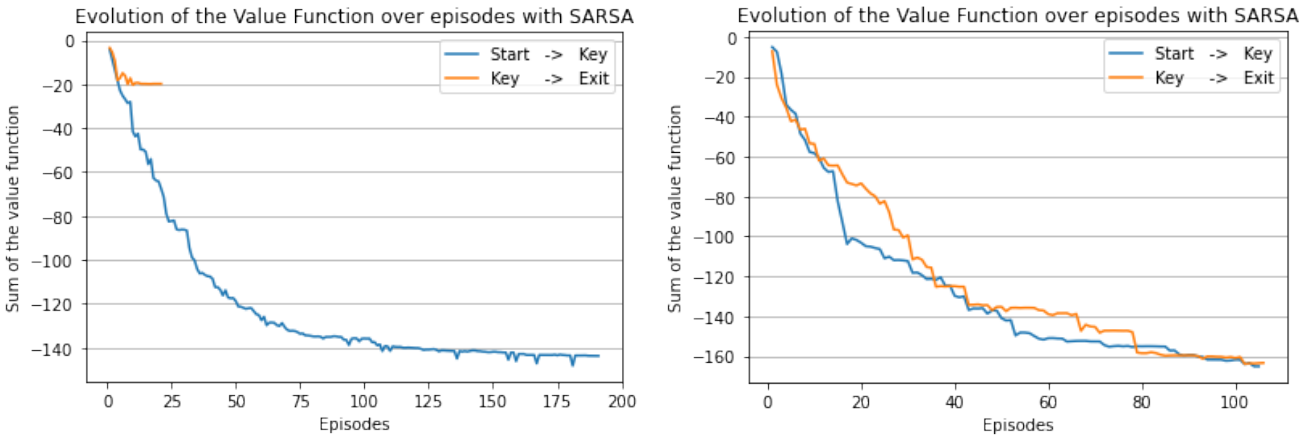

Figure 14: Convergence of Q-Values = 0 initialisation for SARSA algorithm for with different epsilon = {0.1, 0.9}.

Question (3) Influence of a episode dependent ϵ.

The problem B was solved for the value $\epsilon = \frac{1}{k^\delta}$ with $\delta = 0.5$. The convergence of the Q-values can be seen in the figures 15. In each figure, is possible to see the convergence from the state A to state C in the image on the left and from state C to state B on the right

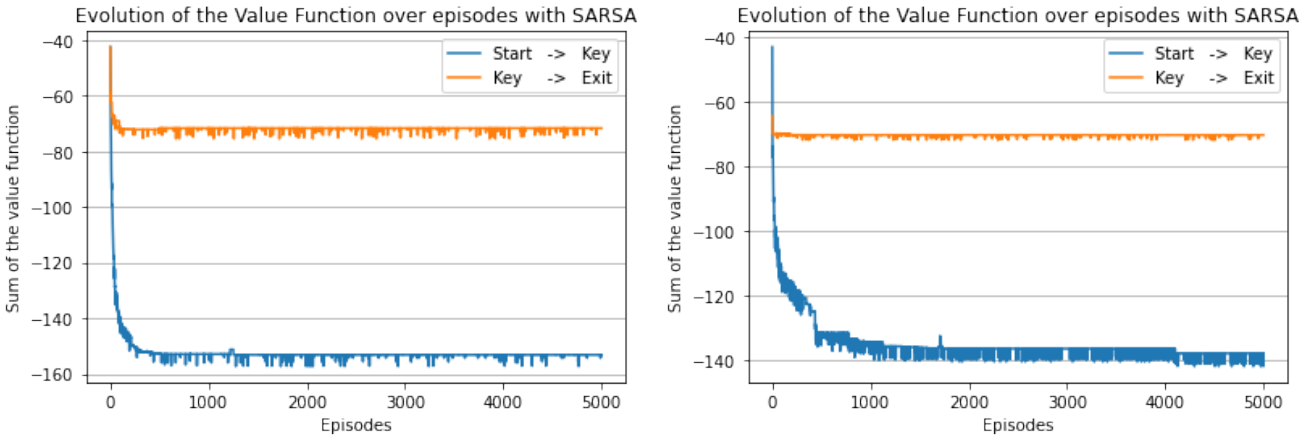

Figure 15: Convergence of Q-Values for $\epsilon = \{\frac{1}{k^{\delta}}\}$ from state A to C and from state C to B. In the left case $(\delta\ , \alpha) = (0.5,\ 0.66)$, and on the right case $(\delta\ , \alpha) = (0.5,\ 0.8)$

According to the figure 15, it is possible to conclude that the convergence was faster and smoother when $\delta < \alpha$.

Question (j) Probabilities of leaving the maze

For both methods the probability of leaving the maze are 100%. The Q-value of the initial state represents the expected value of future states if that action is taken, since they have different meanings, the value might be different.
End of problem 1

Problem 2 - RL with linear function approximators

Presentation of the problem

This problem consist in creating an optimal policy such as a cart could climb a mountain and reach the top of an hill. The problem we have is a Markov Decision Process that we develop :

State Space:

$$S = \{s_1,\ s_2\}; \tag{14}$$

Where s_1 and s_2 are both continuous and are respectively the position and the velocity.
We note S_f is the final position to reach

Action Space:

$$A = \{left\ push,\ no\ push,\ right\ push\} \tag{15}$$

Transition:

$$\begin{cases} p(s'|s,a) = 1/3, & s' \in S \\ p(s'|s,a) = 0, & s' \notin S \end{cases}, \quad s \in S, a \in A \tag{16}$$

$$\tag{17}$$

Rewards:

$$\begin{cases} r(s,a) = 0, & (s',s,a)s' \in Sf \\ r(s,a) = -1, & (s',s,a)s' \notin Sf \end{cases}, s \in S, a \in A \tag{18}$$

Question (b) Implementation of Sarsa(λ) with Fourier Basis and linear function approximation

Pseudo Code of Sarsa(λ) with function approximator and fourier basis

Algorithm 3 SARSA with function approximator and fourier basis

$s_t \leftarrow scaling(env.reset())$
$Weight \leftarrow zero\ vector$
$Z \leftarrow zero\ vector$
$a_t \leftarrow \epsilon - greedy\ policy\ with\ (state,\ Weight\ \epsilon)$
$Q_t \leftarrow Q(s_t, a_t))$
while s $\notin S_f$ **do** $s_{t+1} \leftarrow$step(r_t, a_t)
  $Q_{t+1} \leftarrow Q(s_{t+1}, a_{t+1}, W_t))$
  $\delta \leftarrow r_t + \gamma * Q_{t+1} - Q_t$
  $W_{t+1} = W_t * \delta * Z_a$
  $Z_a update$ ▷ eligibility trace algorithm
  $a_t \leftarrow a_{t+1}$
  $s_t \leftarrow s_{t+1}$
  $Q_t \leftarrow Q_{t+1}$
  if $R > -200$ **then** $\alpha \leftarrow \alpha * 0.7$

Question (c) Description of training process

First there are the basic functions such as : *scale state variables*, *running average* which ensures that the states are scaled in [0,1].
Then there are the RL related function such as *Q function* which apply the Fourier Basis and *policy greedy* which return an action to take given a state and a reward. Also there is the *eligibility trace* which bring the computation of the Z_a. Finally we have the *Sarsa lambda* function which compute the whole SARSA(λ) algorithm. We did not implement any modification of the Stochastic Gradient Descent.

We have took into account the advice given on the paper concerning the learning rate. Each time the average reward is above -200, the learning rate is decreased by 30% step by step. We will analyse after the influence of the learning rate on the average total reward.
We basically trained our model for 200 episodes and for the following values of parameters :

γ	λ	α	$N_{episodes}$	η
0.99	0.89	0.01	200	[[0,1],[1,0],[1,1],[1,2],[2,1],[2,2]]

Question (d) Analysis of training process and optimal policy

(1) Evolution of the episodic reward during training :

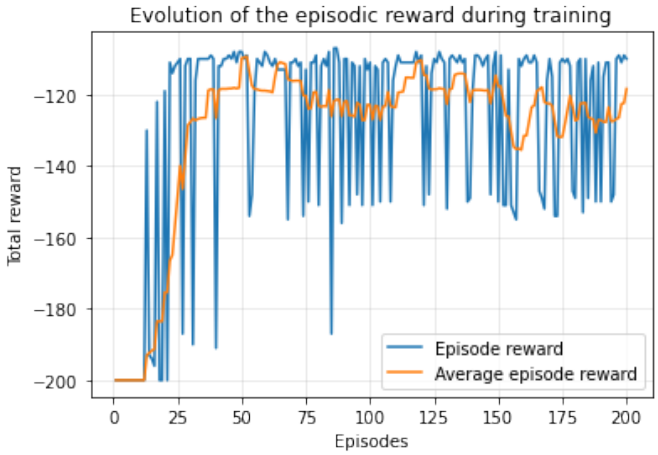

Figure 16: Evolution of the episodic reward during training

We can observe that the goal of reaching a reward above -135 after the first 50th episodes is reached and that it gets better with experience because the final average total reward is -119.4. *Policy achieves an average total reward of -119.4 +/- 4.6 with confidence 95%*

(2) Evolution of the value function of the optimal policy as a function of state space domain :

2D plot version orientation :

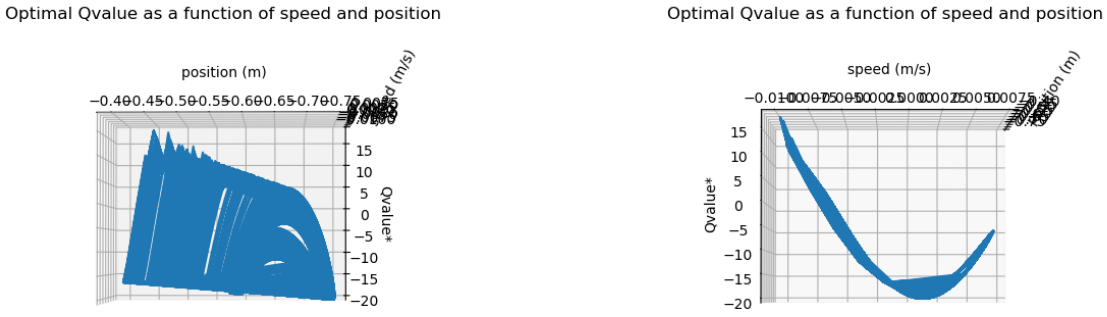

Figure 17: Evolution of the value function of the optimal policy as a function of state space domain (2D)

We can observe that when the cart takes more momentum (so when the position is negative) the Qvalue is increasing. It is logical because the more momentum the cart will have, the more it will reach the cliff.
Concerning the position it is the same, the Qvalue is maximised when the speed in the left direction is important because the cart will acquire more momentum.

We thus obtain the following 3D version :

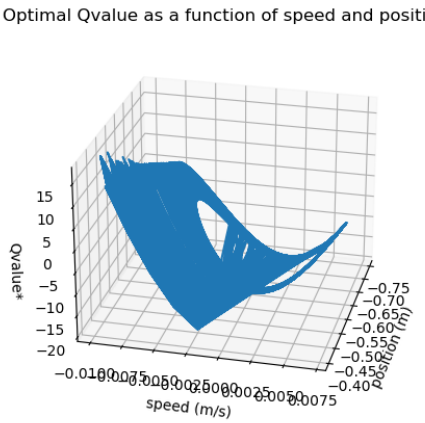

Figure 18: Evolution of the value function of the optimal policy as a function of state space domain (3D)

(3) Evolution of the optimal policy as a function of state space domain :

2D plot version orientation :

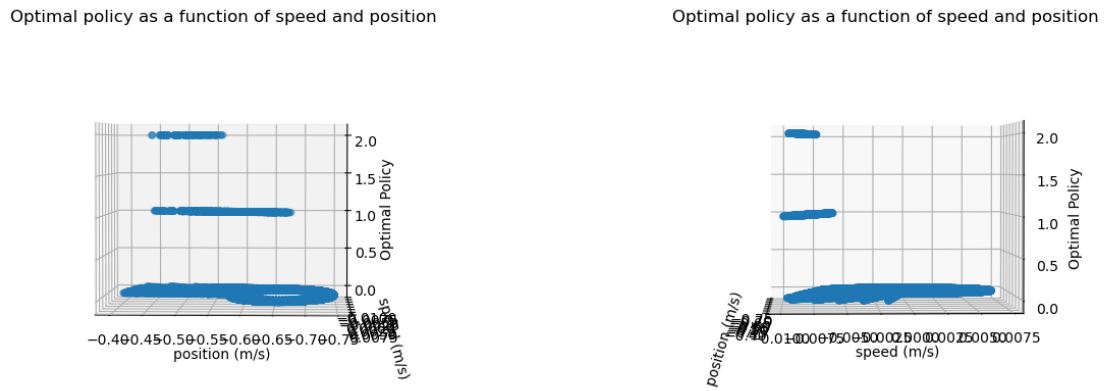

Figure 19: Evolution of the optimal policy as a function of state space domain (2D)

3D plot version orientation :

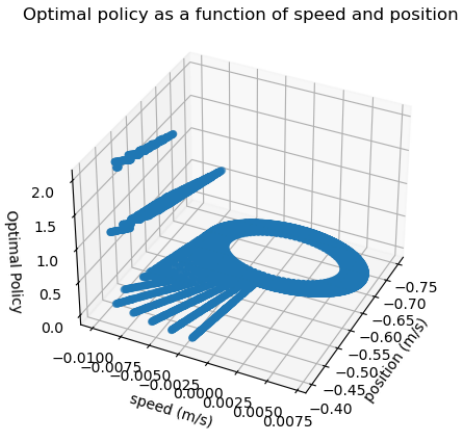

Figure 20: Evolution of the optimal policy as a function of state space domain (3D)

First we can observe an ellipsoid around the bottom zone because the speed is the derivative of the position. When the position is minimal, the speed is maximal and reciprocally.
Secondly, we have seen before that the Qvalue is maximised when the position and speed are minimised because the cart gets more momentum to reach the goal. This is a result that we find also here. The zone where the speed is negative is the zone where the decision are taken because it is where the momentum is high and where we can either push right to have more momentum or do nothing and keep our momentum to reach to top of the cliff.

(4) Vector η :

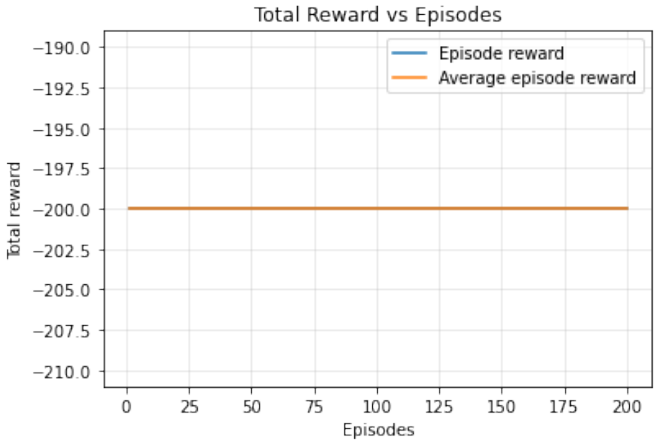

Figure 21: Evolution of the episodic reward during training with a vector η that includes [0,0]

When we add the null value in the vector η we obtain the former plot which means that it is not working.
As the vector η is made to capture the interaction between the state variables and the action variables, putting it null would suggest that the state and action are uncorrelated, which is wrong. So we think that is it why it is not working.

(5) Comparison with an agent that takes random action :

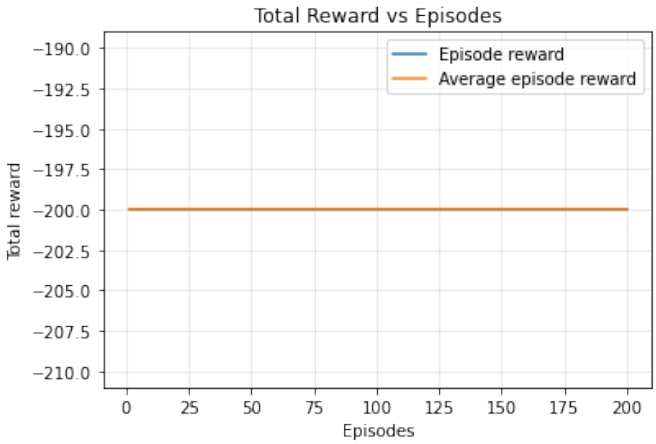

Figure 22: Comparison with an agent that takes random action

When we take a random agent, obviously it does not learn anything so it is not working as we can see on the former plot.

Question (e) Evolution of average total reward

A] Evolution of average total reward as a function of the learning rate α

The values of alpha tested are : [1e-05, 2e-05, 3e-05, 4e-05, 5e-05, 6e-05, 7e-05, 8e-05, 9e-05, 0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008, 0.0009, 0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
Knowing that the value of the learning rate is reduced by scales of 30% each episodes after it has reached -200, the former value are the initial value. The other parameters are fixed as follow :

γ	λ	$N_{episodes}$	η
0.99	0.89	200	[[0,1],[1,0],[1,1],[1,2],[2,1],[2,2]]

The values of alpha we have are very spead, so impossible to visualise in a linear scale. To observe it, when transform the alpha with 20.log10(alpha) which correspond to a dB conversion.

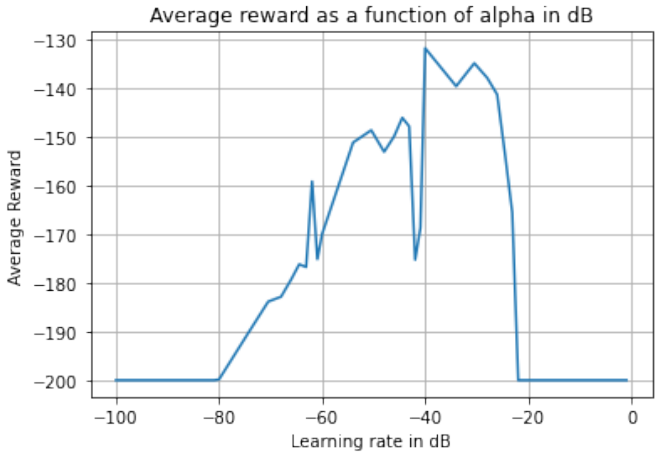

Figure 23: Evolution of average total reward as a function of the logarithm of the learning rate in dB

We can observe that there is a range of value where the average reward is maximised. Around α = [-30 ; -40]dB, we see that the average reward is reaching the goal. So it corresponds approximately to values of : $\alpha = [0.01; 0.031]$

B] Evolution of average total reward as a function of the eligibility trace λ

The eligibility trace is supposed to be clipped between -5 and 5, so we have chosen to observe the evolution of the average reward for values between -2 and 2. $(np.linspace(-2, 2, 60))$
The other parameters are fixed as follow :

γ	α	$N_{episodes}$	η
0.99	0.01	200	[[0,1],[1,0],[1,1],[1,2],[2,1],[2,2]]

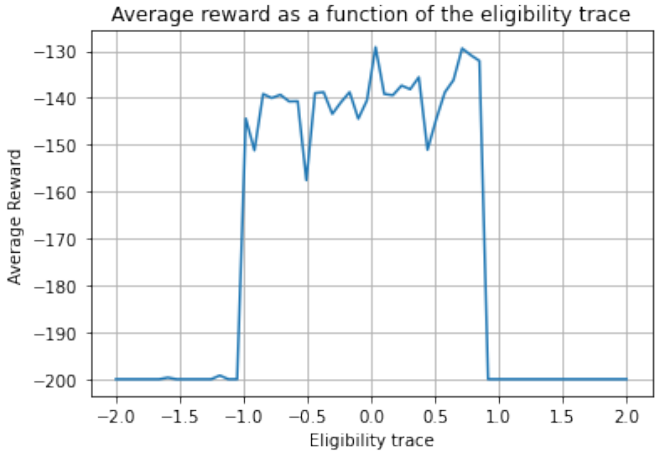

Figure 24: Comparison with an agent that takes random action

We can also observe here that there is a range of value where the eligibility trace maximize the average reward.
The optimal weight vector is the file named "weight.pkl". We have also provide an other weight vector which was obtained with the random agent : (weight_random.pkl").

End of problem 2