Multi-agent systems - implementation of reinforcement learning and Markov decision processes using FrozenLake from gym library

Guy Cohen

Ester Vaknin

Almog Klein

Afeka College of Engineering

April 07, 2022

In this report we will present all the steps we implemented in the code and describe the results we got and draw conclusions.

<u>שלב 1</u>: עליכם לבחור משימה עבור סוכן בודד, שמקיימת תנאי MDP. נא הציגו את המשימה כתהליך קבלת החלטות מרקובי סופי. יש להגדיר באופן מפורש את המצבים (states), פעולות האפשריות בכל מצב, הסתברויות מעבר (transition probabilities) והתשלומים (expected rewards). כמו כן, נא ציינו במפורש את הכללים המיוחדים שיש בסביבה שהגדרתם.

Frozen lake involves crossing a frozen lake from Start(S) to Goal(G) without falling into any Holes(H) by walking over the Frozen(F) lake. The agent may not always move in the intended direction due to the slippery nature of the frozen lake. The agent takes a 1-element vector for actions. The action space is (dir), where dir decides direction to move in which can be:

Action Space	
NO.	**Action**
0	**West**
1	**south**
2	**East**
3	**North**

is_slippery: If True, will move in intended direction with probability of 1/3 else will move in either perpendicular direction with equal probability of 1/3 in both directions.

The observation is a value representing the agent's current position as current_row * nrows + current_col (where both the row and col start at 0).

Reward schedule:

- Reach goal(G): +1 (terminal state)
- Reach hole(H): 0 (terminal state)
- Reach frozen(F): 0

```
SFFF       (S: starting point, safe)
FHFH       (F: frozen surface, safe)
FFFH       (H: hole, fall to your doom)
HFFG       (G: goal, where the frisbee is located)
```

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

$$\Pr\{R_{t+1} = r, S_{t+1} = s' \mid S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t\},$$

$$p(s', r|s, a) = \Pr\{S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a\}.$$

$$p(s'|s, a) = \Pr\{S_{t+1} = s' \mid S_t = s, A_t = a\} = \sum_{r \in \mathcal{R}} p(s', r|s, a),$$

$$r(s, a, s') = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a, S_{t+1} = s'] = \frac{\sum_{r \in \mathcal{R}} r\, p(s', r|s, a)}{p(s'|s, a)}.$$

```
Transition matrix =

 [[0.5  0.25 0.   0.   0.25 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.25 0.25 0.25 0.   0.   0.25 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.25 0.25 0.25 0.   0.   0.25 0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.25 0.5  0.   0.   0.   0.25 0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.25 0.   0.   0.   0.25 0.25 0.   0.   0.25 0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.25 0.   0.   0.25 0.   0.25 0.   0.   0.25 0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.   1.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.25 0.   0.   0.   0.25 0.25 0.   0.   0.25 0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.25 0.   0.   0.25 0.   0.25 0.   0.   0.25 0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.25 0.   0.   0.25 0.   0.25 0.   0.   0.25 0.  ]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   1.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   1.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.   0.25 0.   0.   0.25 0.25 0.25 0.  ]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.25 0.   0.   0.25 0.25 0.25]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   1.  ]]

Reward vector =

 [0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.25 0.  ]
```

<u>שלב 2:</u> עבור מדיניות (policy) π מסוימת (לבחירתכם) נא חישבו ערכי $V^{\pi}(s)$ לכל מצב s באמצעות משוואות בלמן. נא רשמו את המערכת ואת הפתרון. כמו כן, עבור אותה מדיניות π נא חשבו $V^{\pi}(s)$ באמצעות Value Iteration Algorithm.
הוסיפו קטע קוד רלוונטי, ציינו מהם תנאי העצירה ורשמו כמות האיטרציות שהתבצעו. הסיקו מסקנות.

the *value* of a state s under a policy π, denoted $v_\pi(s)$, is the expected return when starting in s and following π thereafter. For MDPs, we can define $v_\pi(s)$ formally as

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \tag{3.10}$$
$$= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big]$$

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
  $\pi(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$

<u>Calculation using Value Iteration Algorithm:</u>

The number of iterations performed, policy and value function:

```
value iteration for  60  iterations
 agent succeeded to reach goal 730117 out of 1000000 Episodes

policy:

[['West(<)' 'North(^)' 'West(<)' 'North(^)']
 ['West(<)' 'West(<)' 'West(<)' 'West(<)']
 ['North(^)' 'south(v)' 'West(<)' 'West(<)']
 ['West(<)' 'East(>)' 'south(v)' 'West(<)']]

value func vector:

[[0.06888624 0.06141117 0.07440763 0.05580502]
 [0.09185097 0.         0.11220727 0.        ]
 [0.14543392 0.24749561 0.29961676 0.        ]
 [0.         0.37993504 0.63901974 0.        ]]
```

stopping cretria and bellman equations:

```
 Stop conditions:
delta < theta : delta = 9.568972661466724e-07

value matrix:

[0.06888703374487129, 0.06664451425610295, 0.06664451425610295, 0.05975509625459354]
[0.03908922434013946, 0.04298816225802422, 0.04074564276925588, 0.06141151468370978 4]
[0.0744078231426156, 0.06882703782948876, 0.07272597574737354, 0.05748714779969938]
[0.03906379537401382, 0.03906379537401382, 0.03348301006088698 4, 0.05580530040445731]
[0.09185133639510051, 0.07118546448064662, 0.06429604647913703, 0.04822116183041738 6]
[0.0, 0.0, 0.0, 0.0]
[0.1122073190157469, 0.08988502867217658, 0.1122073190157469, 0.02232229034357032 7]
[0.0, 0.0, 0.0, 0.0]
[0.07118546448064662, 0.11787885805557752, 0.10180397340685787, 0.14543414797154 1]
[0.15761068697605268, 0.24749571564822925, 0.20386554108354613, 0.13351520323685973]
[0.2996167871005653 3, 0.2659546067272056, 0.22536810360967094, 0.10791086386425411]
[0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0]
[0.18822919590226395, 0.3056864356476808, 0.37993511913857514, 0.2659546067272056]
[0.39557146431985735, 0.6390197689810141, 0.6149242852418212, 0.5371988744168794]
[0.0, 0.0, 0.0, 0.0]
```

At this stage, we presented the Bellman Equation and calculated $V^{\pi}(s)$.

The policy we chose is: $\pi(a|s) = \frac{1}{4}$

We found that the number of iterations we received is 60 iterations and the stopping condition received is 9.56 x 10 ^ -7 < 1 x 10 ^-6. We assume that the algorithm converged to these results due that this game consists of a number of basic and simple operations.

<u>שלב 3</u>: השתמשו ב- Policy Iteration Algorithm למציאת מדיניות אופטימאלית. הציגו את המדיניות האופטימאלית. מהם ערכי V (s) עבור מדיניות אופטימאלית? הוסיפו קטע קוד רלוונטי, ציינו מהם תנאי העצירה ורשמו כמות האיטרציות שהתבצעו. הסיקו מסקנות.

This is an *optimal policy*. Although there may be more than one, we denote all the optimal policies by π_*. They share the same state-value function, called the *optimal state-value function*, denoted v_*, and defined as

$$v_*(s) = \max_{\pi} v_{\pi}(s), \tag{3.13}$$

for all $s \in \mathcal{S}$.

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\quad \Delta \leftarrow 0$
   $\quad$ For each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\big[r + \gamma V(s')\big]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad a \leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
   $\quad$ If $a \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return V and π; else go to 2

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*,$$

```
entered stop criteria after 170 iterations with delta< theta : 8.889676395285O4e-07  <  1e-06

*policy:

[['West(<)' 'North(^)' 'West(<)' 'North(^)']
 ['West(<)' 'West(<)' 'West(<)' 'West(<)']
 ['North(^)' 'South(v)' 'West(<)' 'West(<)']
 ['West(<)' 'East(>)' 'South(v)' 'West(<)']]
value func vector (*policy):

[[0.06888651 0.06141136 0.07440774 0.055805  ]
 [0.09185115 0.         0.11220731 0.        ]
 [0.14543402 0.24749566 0.29961679 0.        ]
 [0.         0.37993506 0.63901975 0.        ]]
```

Stopping criteria is when delta (the change of the value function between every iteration) is smaller than theta = 10^-6. The number of iterations till the stop criteria occurred is 170. We can see that the optimal policy is the same as the policy we got in section 2. This can be explained by the simple game rules and the low number of states and actions. Also, the method we used in section 2 and section 3 converged to optimal policy (maybe local optimum) in relatively low number of iterations. As well as the value functions of both sections are the same probably from the same reasons. Another factor could be the randomness in the environment, every action the agent chooses there is only 33.33% that he will continue to the state that action supposed to lead him (is slippery =True).

<u>שלב 4</u>: עבור אותה מדיניות שנבחרה בשלב 2 חשבו ערכי $V^{\pi}(s)$ לכל מצב s באמצעות למידה תוך שימוש ב- Temporal difference algorithm. הוסיפו קטע קוד רלוונטי, ציינו את ערכי הפרמטרים ותנאי העצירה, ורשמו כמות האיטרציות שהתבצעו. הסיקו מסקנות.

Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0, \forall s \in \mathcal{S}^+$)
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    $A \leftarrow$ action given by π for S
    Take action A; observe reward, R, and next state, S'
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  until S is terminal

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big].$$

```
value func

[[0.00449588 0.00374083 0.00985227 0.00303346]
 [0.00694796 0.         0.01792012 0.        ]
 [0.01710871 0.05485819 0.08427957 0.        ]
 [0.         0.09511005 0.44165715 0.        ]]
```

The number of episodes is 1,500,000. We can see that we got a different value function from the last 2 sections. This one has lower values in general then the ones before although this algorithm has made much more iterations. But the position of the bigger values within this table is closer to the goal state as expected. The max value of this table is 0.44 which is less than half of the reward of an episode, in the only state that can lead to the goal is state. This fact may indicate that the agent has not learned the environment value function well enough. In TD learning the agent needs to play the game to learn, so he needs a lot of playing episodes to learn the value of the states in the game.

<u>שלב 5</u>: השתמשו ב- Temporal difference state-action algorithm למציאת ערכי V (s) לכל מצב s עבור מדיניות אופטימאלית. הציגו טבלה Q התחלתית וטבלה Q אליה התכנס האלגוריתם. רשמו קטע קוד רלוונטי, ציינו את ערכי הפרמטרים ואת כמות הצעדים שנעשו. הסיקו מסקנות.

```
initial q table is:

******* Initial Q-Table *******

 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

to the corresponding algorithm for action values:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big].$$

```
finished with 40114295 Steps for  1500000 episodes



******* Final Q-Table *******

[[0.03734557 0.0444739  0.03721523 0.03738171]
 [0.02620652 0.0293202  0.02664275 0.04329033]
 [0.06616942 0.04220908 0.04099227 0.04014203]
 [0.02793239 0.02612291 0.02664551 0.03556698]
 [0.06262842 0.04443403 0.04565074 0.03224161]
 [0.         0.         0.         0.        ]
 [0.06894727 0.05949653 0.08551704 0.00821178]
 [0.         0.         0.         0.        ]
 [0.06334845 0.06859887 0.07197902 0.13601024]
 [0.1336951  0.21425342 0.13819887 0.10641599]
 [0.25940121 0.18075051 0.14643228 0.12346345]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.15905654 0.24654516 0.26742412 0.22370732]
 [0.38737202 0.55471987 0.47914216 0.4486898 ]
 [0.         0.         0.         0.        ]]
```

Till this section we only used value function to measure the agent learning, and in this section, we will use Q learning. We chose to initialize the (16, 4) Q table with 0 as can be seen in the screenshot. We run the algorithm 1,500,000 episodes, which in them the agent took 40,114,295 steps in the game (26.5 in average for episode). We can see in the final Q table that there are 5 states that have not been learned. Those are the 4 holes in the grid (mentioned in section 1) and the goal state, meaning all the terminal states of the game will have Q table values as we initialize them. Another fact is as we chose bigger state id (meaning closer to the goal state) the values in the table rows are bigger, as expected for the agent learning.

שלב 6: בחלק זה נא השווה תוצאות של Q-learning algorithm עם/ללא eligibility traces. ניתן להציג את ההשוואה בצורה ויזואלית באמצעות גרפים של ההתכנסות. הסיקו מסקנות.

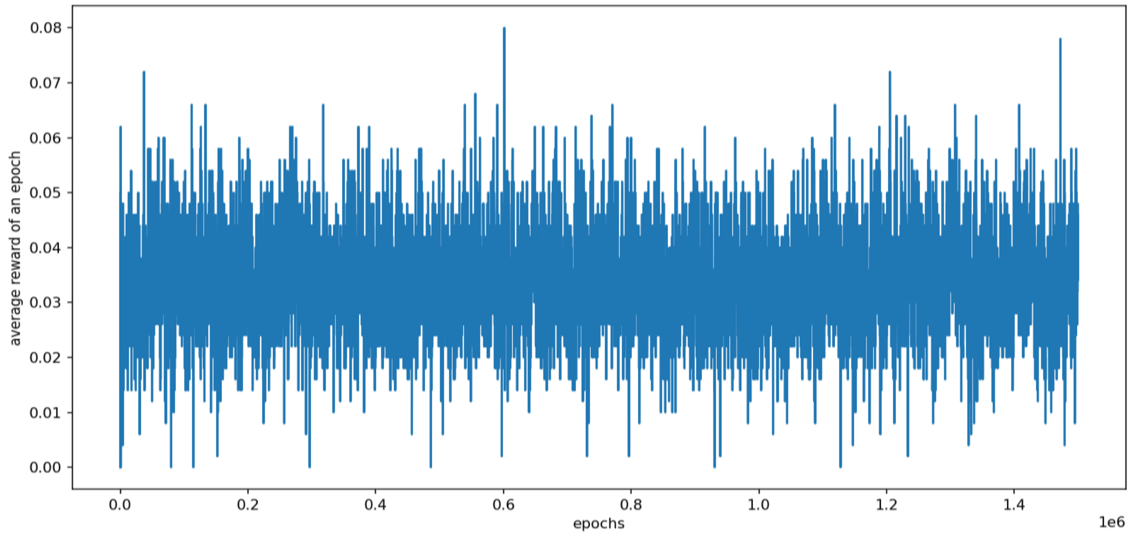
That is, for an episode starting at time step 0 and terminating at step T, for all $s \in \mathcal{S}$:

$$V_{t+1}(s) = V_t(s), \qquad \forall t < T,$$

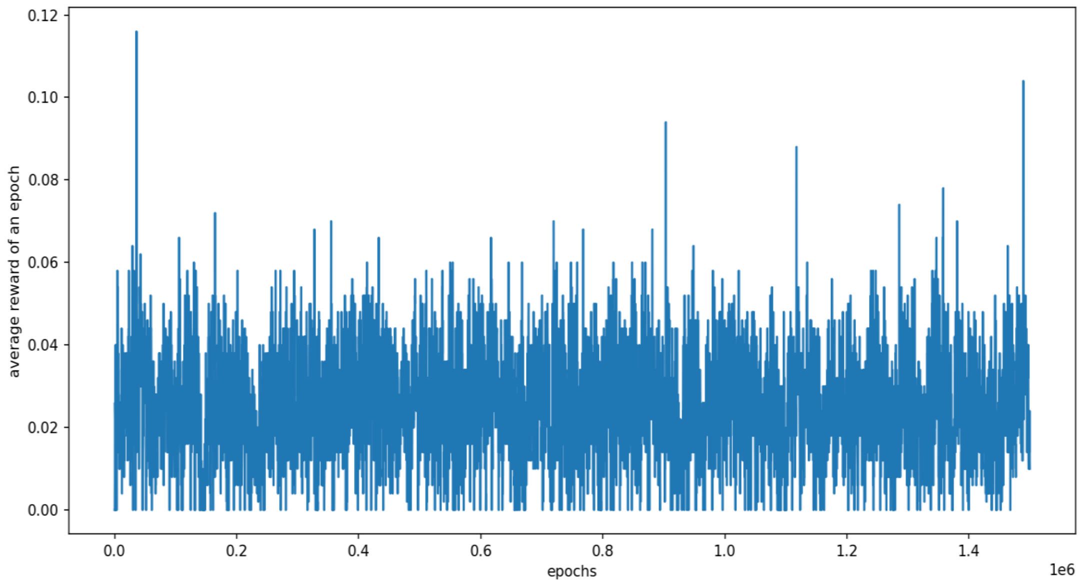
$$V_T(s) = V_{T-1}(s) + \sum_{t=0}^{T-1} \Delta_t(s),$$

$$\begin{aligned} G_t^\lambda &= \sum_{n=1}^{\infty} G_t^{(n)}(1-\lambda_{t+n}) \prod_{i=t+1}^{t+n-1} \lambda_i \\ &= \sum_{k=t+1}^{T-1} G_t^{(k-t)}(1-\lambda_k) \prod_{i=t+1}^{k-1} \lambda_i \;+\; G_t \prod_{i=t+1}^{T-1} \lambda_i. \end{aligned}$$

Learning progress for Q-Learning without eligibly traces



Learning progress for Q-Learning with eligibly traces



```
******* Q-Table *******

[[0.42611206 0.5146688  0.68378067 0.44952703]
 [0.56614517 0.73793606 0.59553851 0.54093271]
 [0.57281976 0.69919344 0.60728218 0.57057674]
 [0.81303196 0.6756013  0.62162979 0.58979903]
 [0.634719   0.78785183 0.55941805 0.54936696]
 [1.         1.         1.         1.        ]
 [0.78182352 0.86066875 0.69403824 0.78128738]
 [1.         1.         1.         1.        ]
 [0.64922002 0.60296072 0.85199473 0.62831113]
 [0.73994143 0.73854687 0.76214952 0.78406708]
 [0.95535285 0.76897928 0.78364447 0.77079921]
 [1.         1.         1.         1.        ]
 [1.         1.         1.         1.        ]
 [0.77830515 0.80920183 0.8069891  1.01877484]
 [0.8214491  1.39593189 0.98599238 1.05696981]
 [1.         1.         1.         1.        ]]
```

```
******* ET-Table *******

[[3.96570448e-075 2.80194362e-006 9.07884771e-001 5.66107658e-093]
 [8.66966374e-230 1.12603384e-001 7.47901769e-135 3.34971326e-086]
 [8.34818567e-205 3.40885196e-005 3.00100081e-077 1.25156559e-260]
 [3.13163796e-008 4.91977510e-114 2.22656386e-195 9.88131292e-324]
 [2.00696597e-007 1.87816660e+000 8.96780076e-046 2.34654535e-068]
 [0.00000000e+000 0.00000000e+000 0.00000000e+000 0.00000000e+000]
 [1.18603504e-085 1.20216726e-012 9.88131292e-324 6.05155199e-086]
 [0.00000000e+000 0.00000000e+000 0.00000000e+000 0.00000000e+000]
 [1.38776219e-267 9.88131292e-324 1.36446600e+000 9.88131292e-324]
 [2.19662118e-292 1.01678023e-301 1.23393555e-206 1.80835464e-008]
 [1.42457656e-014 2.28826792e-207 9.88131292e-324 9.88131292e-324]
 [0.00000000e+000 0.00000000e+000 0.00000000e+000 0.00000000e+000]
 [0.00000000e+000 0.00000000e+000 0.00000000e+000 0.00000000e+000]
 [9.88131292e-324 9.88131292e-324 9.88131292e-324 7.93458946e-071]
 [9.88131292e-324 1.37865940e-034 1.37810377e-210 4.10887751e-210]
 [0.00000000e+000 0.00000000e+000 0.00000000e+000 0.00000000e+000]]
```

The method we used to measure the learning process is every (episodes//STEPS) episodes of the learning we took the existing policy and played 500 episodes in checked how many time the agent won.

The first graph is Q learning without eligibility traces (ET). The second is with ET. We can see that the learning without ET has fewer measures which the average reward is close to 0, than the with ET. The average value of both graphs is the same (maybe a little bit bigger for without ET). but the max values of the graphs are bigger for the with ET learning. The average win percentage is low (around 4%) for both learnings. probably because of the randomness in the environment as mentioned in section 3.