UNIVERSITAT DE BARCELONA

Pràctica 3. Reinforcement Learning

Noah Márquez Vara
Gemma Vallès Fusta

13 desembre 2023

1 OBJECTIUS DE LA PRÀCTICA

L'objectiu principal d'aquesta pràctica és implementar un algorisme de *Reinforcement Learning* per resoldre dos problemes particulars.

Primer tindrem un problema més simple, on donat un tauler de $3x4$ haurem de trobar el millor camí fins a la casella objectiu tenint en compte que hi ha una casella inaccessible i definint la recompensa que té cada casella.

En l'altre problema tenim un taulell d'escacs amb dues peces blanques que podem moure, el rei i la torre, i un rei negre immòbil. L'objectiu consistirà en trobar el camí fins a l'escac i mat movent les peces blanques.

2 ALGORISME IMPLEMENTAT (Q-LEARNING)

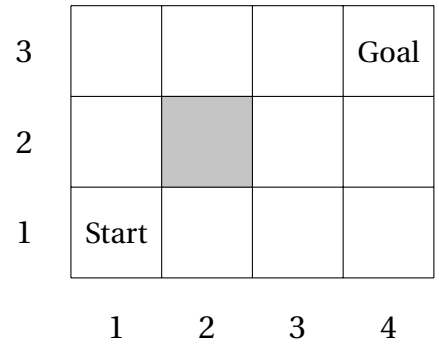
L'algorisme Q-learning és un algorisme de reforç actiu que es basa en l'experiència dels valors obtinguts per a esbrinar el valor que té fer una acció en particular. Es pot aplicar en entorns estocàstics sense requerir adaptacions.

Consisteix en buscar aproximacions successives als valors òptims, cada vegada que tenim una nova mostra la afegim a la nostra taula tenint també en compte els valors anteriors. Tenim diversos paràmetres que permeten variar la importància que l'algorisme li dona a l'informació antiga, les recompenses futures i a explorar o explotar.

Com major sigui el valor de la taxa d'aprenentatge, més importància li donarà a l'informació nova respecte l'antiga. Com major sigui el factor de descompte més importància li donarà a les recompenses futures. També hem d'establir la probabilitat d'escollir explorar en lloc d'explotar la informació ja coneguda.

3 QÜESTIONS PLANTEJADES

1. We will start by tackling the simple problem seen in class of finding a path from start to goal in the following scenario:



(a) **Implement the Q-learning algorithm to find the optimal path considering a reward of -1 everywhere except for the goal, with reward 100.**

i. **Print the first, two intermediate and the final Q-table. What sequence of actions do you obtain?**

```
Initial Q-table:
[[[ 62.171        0.           0.          62.11696321]
  [  0.           0.          54.9537143   70.18276666]
  [ 79.09983829   0.          -0.999999     0.        ]]

 [[ 70.19        54.9538964    0.           0.        ]
  [  0.           0.           0.           0.        ]
  [ 89.          -0.99999      0.           0.        ]]

 [[ 79.09999894  62.15989287   0.          79.1       ]
  [ 89.           0.          70.18998474  88.99999933]
  [100.          79.09901922  79.09199092   0.        ]]

 [[  0.          70.18999999   0.          89.        ]
  [  0.          79.1         79.09999917 100.        ]
  [  0.           0.           0.           0.        ]]]

Q-table after 9 episodes:
[[[-1.          0.          0.         62.05237045]
  [ 0.          0.         53.93534042 70.16260129]
  [79.09811292  0.         62.0481708   0.        ]]

 [[-0.9999999  42.23687802  0.          0.        ]
  [ 0.          0.          0.          0.        ]
  [88.99973028 69.97773333  0.          0.        ]]

 [[-0.999      -0.999       0.         -0.99      ]
  [-0.99        0.          0.         71.91      ]
  [99.99999    78.2021448   0.          0.        ]]

 [[ 0.         -0.9         0.         71.901     ]
  [ 0.         -0.99       -0.9        99.        ]
  [ 0.          0.          0.          0.        ]]]

Q-table after 16 episodes:
[[[ 38.98410887   0.           0.          61.79692447]
  [  0.           0.          54.05819975  70.02832971]
  [ 79.08476389   0.          60.66154219   0.        ]]

 [[ 42.079935    44.16457557   0.           0.        ]
  [  0.           0.           0.           0.        ]
  [ 89.          68.95585105   0.           0.        ]]

 [[ 57.23991     48.17180337   0.          -0.99      ]
  [ 79.191        0.           0.          88.98290991]
  [100.          79.09102141  78.60561292   0.        ]]
```

```
 [[  0.          -0.9          0.          86.4801    ]
  [  0.          71.07715703  -0.9         99.9       ]
  [  0.           0.           0.          0.        ]]]

Final Q-table:
[[[ 53.85836863   0.           0.          62.171     ]
  [  0.           0.          54.95389998  70.19      ]
  [ 79.1          0.          62.17099985   0.        ]]

 [[ 61.63972925  54.93851278   0.           0.        ]
  [  0.           0.           0.          0.        ]
  [ 89.          70.18998766   0.          0.        ]]

 [[ 57.23991     48.17180337   0.          78.29633141]
  [ 88.88571      0.           0.          88.99999983]
  [100.          79.0999991   79.099992     0.        ]]

 [[  0.          -0.9          0.          86.4801    ]
  [  0.          71.07715703  -0.9         99.999     ]
  [  0.           0.           0.          0.        ]]]

Visual Q-table:
right      | right      | right      | N/A        |
up         | N/A        | up         | up         |
up         | right      | up         | up         |
```

[1]

La seqüència d'accions obtinguda és la següent: ['up', 'up', 'right', 'right', 'right'].

ii. **After trying for a bit, what is your parameter choice for alpha, gamma and epsilon? Why?**
Per a escollir els paràmetres alpha, gamma i epsilon principalment hem tingut en compte dos factors: el temps d'execució i la precisió, és a dir, que trobi el camí correcte sempre i en el menor temps possible.

Per a trobar els paràmetres que compleixen amb aquests dos requisits primer hem analitzat quins paràmetres reduïen més el temps d'execució i després ens hem centrat en trobar una combinació que fos precisa.

Per a això primer hem anat variant cada pàrametre individualment per veure com afectava això al seu temps d'execució:

[1] Hem afegit una Q-table que tradueix els valors a les accions òptimes en cada estat per tal de fer-ho més fàcil d'interpretar a simple vista.

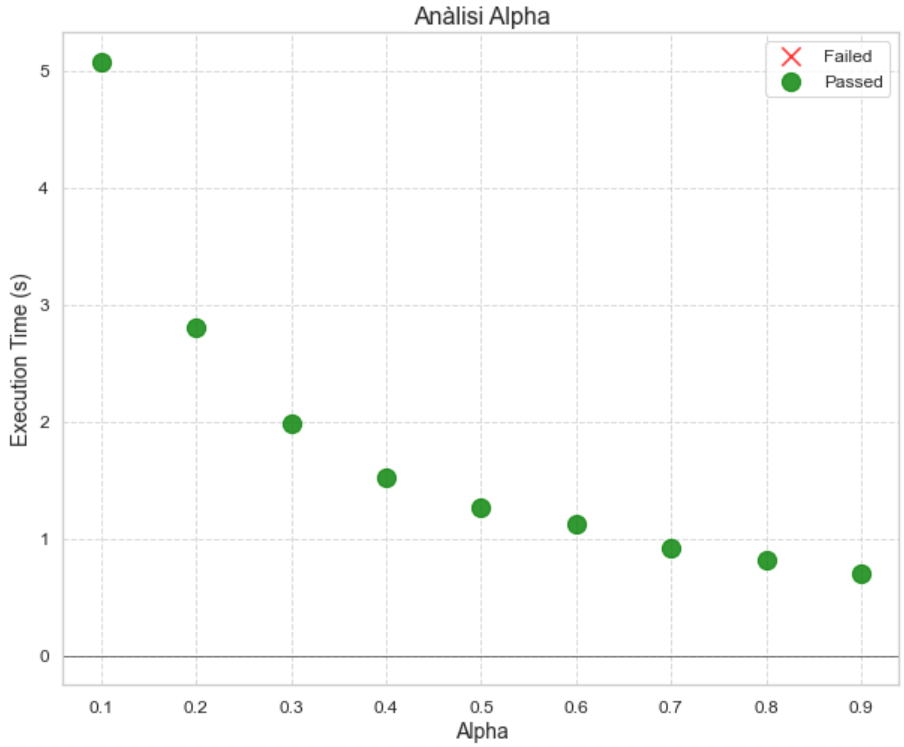


Figura 3.1: Alpha (Mitjana de 10.000 execucions)

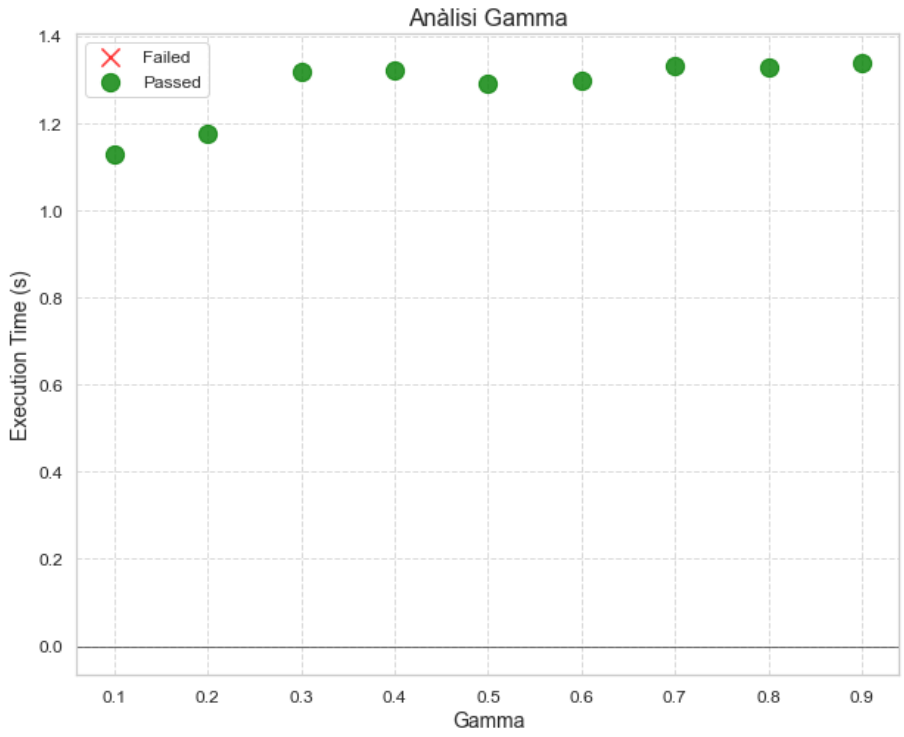


Figura 3.2: Gamma (Mitjana de 10.000 execucions)

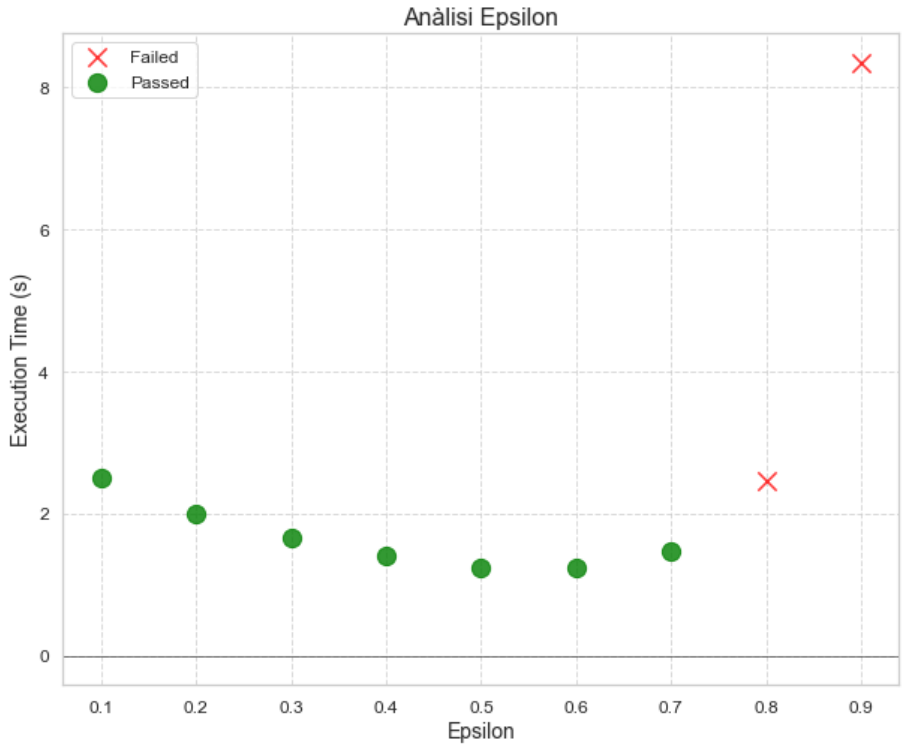


Figura 3.3: Epsilon (Mitjana de 10.000 execucions)

Un cop obtingudes aquestes dades, podem analitzar quins valors són millors pel nostre objectiu.

Podem veure que en el cas de *gamma* els temps quasi no varien i en el cas de l'*epsilon* entre 0,3 i 0,7 tampoc. En canvi, en el cas d'*alpha*, podem observar que com més gran més es redueix el temps d'execució, de forma considerable.

Ara que ja hem vist quins valors disminueixen més el temps d'execució, falta veure quins d'aquests valors ho fan sense error. Com ja hem vist, alts valors d'*epsilon* tendeixen a fallar, això pot ser degut a que al explorar poc acabem convergint abans d'hora.

Finalment, provant uns quants valors més hem concluït que els nostres paràmetres ideals són *alfa* = 0,9, *gamma* = 0,9 i *epsilon* = 0,4. Per comprovar que no fallés, l'hem executat 100.000 vegades amb èxit.

iii. **How do you judge the convergence of the algorithm? How long does it take to converge?**
Per a comprovar si l'algorisme ha convergit, mirem si la taula actual i l'anterior són iguals segons una tolerància. Amb els paràmetres anteriors, l'algorisme ens tarda una mitjana de 28,29 episodis en convergir.

(b) **Try implementing the more accurate reward given by:**

3	-3	-2	-1	100
2	-4		-2	-1
1	-5	-4	-3	-2
	1	2	3	4

i. **Answer the questions of the previous section for this case.**

Print the first, two intermediate and the final Q-table. What sequence of actions do you obtain?

```
Initial Q-table:
[[[ 53.85836863   0.           0.          62.171     ]
  [  0.           0.          54.95389998  70.19      ]
  [ 79.1          0.          62.17099985   0.        ]]

 [[ 61.63972925  54.93851278   0.           0.        ]
  [  0.           0.           0.           0.        ]
  [ 89.          70.18998766   0.           0.        ]]

 [[ 57.23991     48.17180337   0.          78.29633141]
  [ 88.88571      0.           0.          88.99999983]
```

```
  [100.          79.0999991   79.099992     0.        ]]

 [[  0.          -0.9          0.          86.4801    ]
  [  0.          71.07715703  -0.9         99.999     ]
  [  0.           0.           0.          0.        ]]]

Q-table after 9 episodes:
[[[56.55323848  0.          0.         -3.99996   ]
  [ 0.          0.         19.69936681 -2.9997    ]
  [-1.99998     0.         -3.96        0.        ]]

 [[67.28841905 45.36516604  0.          0.        ]
  [ 0.          0.          0.          0.        ]
  [71.901      -2.997       0.          0.        ]]

 [[78.04630604 56.53071218  0.         78.09889542]
  [88.9999416   0.         67.27536199 79.2       ]
  [90.         -1.8        77.7048822   0.        ]]

 [[ 0.         64.82074039  0.         88.9999811 ]
  [ 0.         77.30091195 78.08222409 99.999999  ]
  [ 0.          0.          0.          0.        ]]]

Q-table after 16 episodes:
[[[ 56.56061014   0.           0.          31.55266073]
  [  0.           0.          45.53208729  -2.99997   ]
  [ 56.239812     0.          -3.96         0.        ]]

 [[ 67.28999685  45.89620897   0.           0.        ]
  [  0.           0.           0.           0.        ]
  [ 79.1901      -2.997        0.           0.        ]]

 [[ 78.09999456  56.53071218   0.          78.09999966]
  [ 89.           0.          67.27536199  87.21      ]
  [ 99.9         -1.8         77.7048822    0.        ]]

 [[  0.          67.26527748   0.          89.        ]
  [  0.          78.09200911  78.0999982  100.        ]
  [  0.           0.           0.          0.        ]]]

Final Q-table:
[[[ 56.561        0.           0.          37.31425023]
  [  0.           0.          45.90489627  42.55425072]
  [ 67.9679622    0.          -3.96         0.        ]]

 [[ 67.29        45.90489912   0.           0.        ]
  [  0.           0.           0.           0.        ]
  [ 88.9811991   -2.997        0.           0.        ]]

 [[ 78.1         56.55797122   0.          78.1       ]
```

```
 [ 89.          0.          67.28999985  88.74       ]
 [ 99.999      77.13420642  77.7048822    0.         ]]

[[  0.         67.28975277   0.          89.         ]
 [  0.         78.09999992  78.1        100.         ]
 [  0.          0.           0.          0.         ]]]

Visual Q-table:

right      | right      | right      | N/A        |
down       | N/A        | right      | up         |
right      | right      | up         | up         |
```

La seqüència d'accions obtinguda és la següent: ['up', 'up', 'right', 'right', 'right'].

After trying for a bit, what is your parameter choice for alpha, gamma and epsilon? Why?
Hem tornat a utilitzar els mateixos paràmetres d'abans ja que l'algorisme és més ràpid però segueix funcionant correctament amb aquests.

How do you judge the convergence of the algorithm? How long does it take to converge?
Per a comprovar si l'algorisme ha convergit, mirem si la taula actual i l'anterior són iguals segons una tolerància. Amb els paràmetres anteriors, l'algorisme ens tarda una mitjana de 30 episodis en convergir.

ii. **What is the effect of the new reward function on performance?**
Podem veure que el temps d'execució és bastant més baix.

iii. **How does this relate to the search algorithms studied in P1? Could you apply one of those in this case?**
La relació que té és que aquest algorisme és similar al A^* ja que en els dos casos s'està buscant el camí més curt entre dos punts. És podria aplicar l'algorisme de la pràctica 1 i fer servir la Q-table com a heurística.

(c) **The main novelty in RL algorithms with respect to the search algorithms in P1 is that they can be applied in stochastic environments, where the agent doesn't fully determine the outcome of its actions.**

i. **Drunken sailor. Your agent is now a drunken sailor trying to get to bed after a good share of whiskey and shanties: their legs don't seem to obey them all the time. Introduce stochasticity (= randomness) by enforcing that only 99% of the steps intended by the sailor are actually taken, the rest leading randomly in any other possible direction.**

ii. **Use at least one of the two rewards proposed:**
Utilitzarem la segona recompensa.

1. **What is your parameter choice? Why?**
Si ens basem en un equilibri entre error i temps, els paràmetres que hem escollit són *alfa* = 0.8, *gamma* = 0.9 i *epsilon* = 0.5, ja que és força ràpid, convergeix en pocs episodis i té un error de 1.85%. Tot i que hi ha altres combinacions de paràmetres que són més precisos, hem comprovat que aquests tarden significativament més del que redueixen l'error al trobar el camí correcte.
2. **Assuming the sailor is in a state that allows learning: how many drunken nights are necessary for them to master the perilous path to bed? Compare to**

the previous, deterministic scenario. Amb els paràmetres anteriors, tardaria unes 45 nits. Veiem que aquest valor és més alt que en el cas determinista, però també cal tenir en compte que si vulguèssim més precisió aquest nombre pujaria bastant, mentres que en el cas determinista tenim 100% de precisió amb menys episodis.

3. **What is the optimal path found? If we watched the sailor try to follow it, would they always follow the same path?**
   El camí òptim és: [(0, 0), (1, 0), (2, 0), (3, 0), (3, 1), (3, 2)]. Tal i com ens indica l'enunciat, en un 1% de les vegades el navegant no seguirà el camí.
4. **Could we apply one of the algorithms in P1 here? Why?** En aquest cas no és poden aplicar els algorismes de la pràctica 1, ja que no és possible aplicar-los en problemes no deterministes.

2. Now, let's move back to the chess scenario, namely the first board configuration of P1. Remember that we have the black king, the white king and one white rook, and that only whites move. Remember to provide the first, two intermediate and the final Q-table in every case.

(a) **Adapt your Q-learning implementation to find the optimal path to a check mate considering a reward of -1 everywhere except for the goal (check mate for the whites), with reward 100.**

i. **What sequence of actions do you obtain?**
([7, 0, 2], [7, 4, 6]) -> [[7, 0, 2], [6, 3, 6]] -> [[7, 0, 2], [5, 2, 6]] -> [[0, 0, 2], [5, 2, 6]] -> [[0, 0, 2], [4, 3, 6]] -> [[0, 0, 2], [3, 4, 6]] -> [[0, 0, 2], [2, 4, 6]]. (6 steps)

ii. **After trying for a bit, what is your parameter choice for alpha, gamma and epsilon? Why?**
Després de provar vàries combinacions de paràmetres d'alpha, gamma i epsilon, hem considerat que els millors són:

- $\alpha = 0.5$ (*learning rate*)
- $\gamma = 0.8$ (*discount factor*)
- $\epsilon = 0.85$ (*exploration rate*

Hem pogut comprovar que amb aquests valors, l'algorisme triga menys a convergir (uns 4500 episodis) tot mantenint els bons resultats. És a dir, tot i estar *explotant* el 85% de les vegades la *q_table*, i amb un *learning rate* de 0.5 (factor que determina quina importancia donem a l'aprenentatge futur), som capaços d'aconseguir una seqüència d'accions de 6 passos, que és el mínim d'accions necessàries per arribar a un *check-mate* donat l'estat inicial del taulell proporcionat a la P1.

iii. **How do you judge convergence of the algorithm? How long does it take to converge?**
Per tractar la convergència en el nostre *Q_learning* es fa servir un llindar de convergència. Aquest llindar és un valor preestablert que determina quan l'algoritme ha arribat a un punt on els valors de la *Q Table* ja no experimenten canvis significatius.

La convergència es mesura per episodi, sumant la diferència absoluta entre els valors antics i nous de la *Q Table* per a cada acció realitzada en un estat. Si aquesta suma total de diferències és menor que el llindar de convergència, es considera que l'algoritme ha convergit.

El temps necessari per a la convergència depèn dels valors dels paràmetres escollits. En el nostre cas la convergència triga aproximadament uns 4500 episodis (uns

2 minuts al nostre ordinador), però això pot variar.

Aquí mostrem un fragment del codi que resumeix com es tracta la convergència:

```
convergence_threshold = 0.01
...
for episode in range(self.episode):
    total_difference = 0
    ...
    for each action in episode:
        old_value = self.qTable[state][action]
        ...
        # update qTable and calculate total
            difference
        total_difference += abs(self.qTable[
            state][action] - old_value)
    ...
    if total_difference < convergence_threshold
        :
        # Convergence achieved
        ...
```

Aquest fragment destaca la part del codi on es calcula la diferència total i es compara amb el llindar de convergència per a determinar si l'algoritme ha convergit.

(b) **Try now with a more sensible reward function adapted from the heuristic used for the A* search:**

i. **Answer the questions of the previous section for this case.**

i. **What sequence of actions do you obtain?**
([7, 0, 2], [7, 4, 6]) -> [[7, 0, 2], [6, 5, 6]] -> [[0, 0, 2], [6, 5, 6]] -> [[0, 0, 2], [5, 6, 6]] -> [[0, 0, 2], [4, 5, 6]] -> [[0, 0, 2], [3, 5, 6]] -> [[0, 0, 2], [2, 4, 6]] (6 steps).

ii. **After trying for a bit, what is your parameter choice for alpha, gamma and epsilon? Why?**
Després de provar vàries combinacions de paràmetres d'alpha, gamma i epsilon, hem considerat que els millors són:

- $\alpha = 0.5$ (*learning rate*)
- $\gamma = 0.9$ (*discount factor*)
- $\epsilon = 0.95$ (*exploration rate*

Hem pogut comprovar que amb aquests valors, l'algorisme triga menys a convergir (uns 1800 episodis) tot mantenint els bons resultats. És a dir, tot i estar *explotant* el 95% de les vegades la *q_table*, i amb un *learning rate* de 0.5 (factor que determina quina importancia donem a l'aprenentatge futur), som capaços d'aconseguir una seqüència d'accions de 6 passos, que és el mínim d'accions necessàries per arribar a un *check-mate* donat l'estat inicial del taulell proporcionat a la P1.

iii. **How do you judge convergence of the algorithm? How long does it take to converge?**
Per tal tractar la convergència en aquest cas en el que estem fent servir l'heurística utilitzada per l'algoritme de l'A^*, continuem amb la mateixa idea que sense l'heurística. Tenim un llindar de convergència i per cada episodi comparem les seves *Q Tables* per tal de decidir si s'ha convergit o no.

Aquí mostrem un fragment del codi que resumeix com es tracta l'ús de l'heurística:

```
def feelBack(self, state, action,
    heuristic=False):
    '''
    Return the next state and current
       reward according to State and
       action
    '''
    nextstate = action
    if self.isCheckMate(state):
        reward = 100
    else:
        if heuristic:
            # Use heuristic function to
               calculate a penalty based
                on the distance
            # from the goal state. The
               farther away, the larger
               the penalty.
            reward = -1 * self.h(
               nextstate)
        else:
            reward = -1
    return nextstate, reward
```

Aquest fragment destaca la part del codi on es decideix que si estem fent servir l'heurística, en cas de no trobar-nos en un *check-mate*, la recompensa es $-1 \cdot$ heurística.

ii. **What is the effect of the new reward function on performance?**
La diferència en rendiment és bastant notòria. Observem com ara en uns 1800 episodis aconseguim que l'algorisme arribi a convergència. A més, es rebaixa el temps d'execució dels 2 minuts a uns 30 segons aproximadament. Tot això mantenint els bons resultats.

(c) **Drunken sailor. On their way to bed, our drunken sailor sees a chessboard on a table, coincidentally configured as in the previous section. They have seen the captain play with the first mate and want to give it a try, but only have a rudimentary knowledge of the rules (they know how each piece moves and what is a check mate, but not that blacks move as well).**

i. **Introduce stochasticity (= randomness) by enforcing that only a given percentage of the moves intended by the sailor are actually taken, the rest taken randomly from all other possibilities**
Per tal d'introduir aleatorietat, hem afegit una variable *stochasticity_rate*, que el que fa és que de totes les accions que intenti fer el jugador, un percentatge d'elles acabin sent aleatòries.

Per tal d'aconseguir això, hem modificat la nostra funció que escolleix l'acció del jugador:

```
...
if drunk_sailor:
```

```
            if np.random.uniform() < stochasticity_rate
                :
                 # Intended action - choose according to
                     the Q-learning policy
                 if np.random.uniform() < self.epsilon:
                     stateActionList = self.qTable[str(
                        state)]
                     max_list = []
                     max_value = max(stateActionList.
                        values())
                     for k, v in stateActionList.items()
                         :
                          if v == max_value:
                              max_list.append(k)
                     action = np.random.choice(max_list)
                 else:
                     choicelist = []
                     for nextstate in nextstatelist:
                          choicelist.append(str(nextstate
                             ))
                     action = np.random.choice(
                        choicelist)
            else:
                 # Random action - not the intended one
                 choicelist = [str(nextstate) for
                    nextstate in nextstatelist if str(
                    nextstate) != str(state)]
                 action = np.random.choice(choicelist)

            return ast.literal_eval(action)
            ...
```

Veiem que si estem en el cas del *drunken sailor* generem un nombre aleatori, si el nombre obtingut està per sota del valor donat per la variable *stochasticity_rate*, llavors escollirem una acció segons explotació o exploració. Però si en canvi el nombre generat està per sobre del valor de la variable, llavors escollirem una acció de manera aleatoria.

Si per exemple tenim un valor de *stochasticity_rate* = 0.8, voldrà dir que un 20% dels cops escollirà fer una acció aleatòria en lloc de la que tenia pensada.

ii. **Use any reward you prefer:**

1. **What is your parameter choice? Why?**
   Per tal de seguir amb el bon rendiment que ens donava l'heurística, per aquest apartat l'hem continuat fent servir per tal de generar les recompenses.

   Després de provar vàries combinacions de paràmetres d'alpha, gamma i epsilon, hem considerat que els millors són:

   - $\alpha = 0.5$ (*learning rate*)
   - $\gamma = 0.9$ (*discount factor*)
   - $\epsilon = 0.95$ (*exploration rate*
   - stochasticity_rate = 0.95 (voldrà dir que el 5% de les accions seran aleatòries)

Hem pogut comprovar que amb aquests valors, l'algorisme triga menys a convergir (uns 1800 episodis) tot mantenint els bons resultats. És a dir, tot i estar generant aleatòriament les accions durant un 5% dels cops, aconseguim un resultat òptim, amb 6 accions per tal d'arribar a un *check-mate*:

[[7, 0, 2], [7, 4, 6]] -> [[0, 0, 2], [7, 4, 6]] -> [[0, 0, 2], [6, 4, 6]] -> [[0, 0, 2], [5, 4, 6]] -> [[0, 0, 2], [4, 4, 6]] -> [[0, 0, 2], [3, 4, 6]] -> [[0, 0, 2], [2, 4, 6]] (6 steps).

Hem fet vàries proves, i hem comprovat que si per exemple posem un stochasticity_rate = 0.4 (voldrà dir que el 60% de les accions seran aleatòries), la convergència triga aproximadament 1 minut o bé uns 3700 episodis.

2. **Assuming our obsessive sailor is in a state that allows learning: how many games do they have to play before they are satisfied that they have found the best strategy and can go to bed? Compare to the previous, deterministic scenario.**
   Tot dependrà del valor d'stochasticity_rate que assignem al jugador, ja que si posem un valor més baix (voldrà dir més accions aleatòries, a.k.a. més borratxo), la convergència trigarà més que no pas si posem un valor més alt, evitant molt més les accions aleatòries.

   Com hem comentat abans, com més aleatorietat, més trigarà la convergència, comparat amb l'escenari anterior. De mateixa manera, si tenim menys aleatorietat, la convergència trigarà aproximadament el mateix que amb el cas de l'apartat anterior.
3. **What is the optimal path found? If we watched the sailor try to follow it, would they always follow the same path?**
   El camí a seguir és: [[7, 0, 2], [7, 4, 6]] -> [[0, 0, 2], [7, 4, 6]] -> [[0, 0, 2], [6, 4, 6]] -> [[0, 0, 2], [5, 4, 6]] -> [[0, 0, 2], [4, 4, 6]] -> [[0, 0, 2], [3, 4, 6]] -> [[0, 0, 2], [2, 4, 6]].

   Està clar però, que si el jugador intenta seguir aquest camí, donada una certa aleatorietat, podria acabar escollint una acció que no toca (és a dir, una acció no recomendada per la *Q Table*. O bé una acció no tan profitosa com seria la que proporciona aquest camí.

(d) **Compare the application of Q-learning in this chess scenario with that of the grid of exercise 1. How do the two scenarios differ? How does that translate into your results?**
L'aplicació del *Q-learning* tant en el problema de la graella com en l'escenari dels escacs comparteix els mateixos principis fonamentals d'aprenentatge per reforç, però s'apliquen a entorns amb diferents complexitats i normes.
A continuació es presenten les diferències clau i com es tradueixen en els resultats:

a) **Complexitat de l'Espai d'Estats**:
   - La graella és un entorn simple i finit amb un nombre petit i definit d'estats, la qual cosa fa que l'espai d'estats sigui relativament fàcil de gestionar.
   - El problema dels escacs té un espai d'estats significativament més gran a causa de la varietat de posicions i moviments possibles per a cada peça, la qual cosa fa que el problema sigui més complex i el procés d'aprenentatge potencialment més lent a causa del major nombre d'estats possibles a avaluar.

b) **Espai d'Accions**:
   - En el problema de la graella, les accions són simplement direccions de moviment (amunt, avall, esquerra, dreta), que són consistents en tot l'espai d'estats.

- En els escacs, l'espai d'accions és més complex a causa de les normes que governen com es pot moure cada peça. Aquesta variabilitat requereix un mètode més sofisticat per determinar la validesa i les conseqüències de les accions.

c) **Estructura de Recompensa**:
   - El problema de la graella probablement té una estructura de recompensa més simple, amb un únic estat objectiu que produeix una recompensa positiva i totes les altres transicions possiblement tenint una recompensa neutra o lleugerament negativa per fomentar l'exploració.
   - L'escenari dels escacs implica una estructura de recompensa més dinàmica. Hi ha recompenses o penalitzacions intermèdies (com ara penalitzacions basades en l'heurística per estar més lluny d'un escac i mat) i una recompensa final més gran per aconseguir el mat.

d) **Transicions**:
   - En la graella, les transicions són deterministes: si et mous en una direcció, aniràs en aquella direcció llevat que siguis bloquejat per un estat inaccessible.
   - Les transicions en els escacs poden ser més complexes a causa de les interaccions entre les peces. Els moviments de l'oponent poden afectar la dinàmica de les transicions, introduint un element estocàstic encara que el model en si mateix pugui ser determinista.

e) **Estats Terminals**:
   - En la graella, l'estat terminal és la posició objectiu.
   - En els escacs, els estats terminals inclouen posicions de mat o taules, que són més complexes d'identificar algorítmicament.

f) **Variació *Drunken Sailor***:
   - Per a tots dos problemes, el concepte del *Drunken Sailor* introdueix estocasticitat, però els seus efectes són més pronunciats en l'escenari dels escacs a causa de l'espai d'estats més gran i les accions complexes.

Aquestes diferències es tradueixen en els resultats de la següent manera:

- **Convergència**: El problema de la graella convergirà més ràpidament cap a una política òptima a causa de la seva simplicitat.
- **Complexitat de la Política Òptima**: L'escenari dels escacs requereix una política més matissada per manejar l'espai d'estats més gran i les interaccions complexes.
- **Estabilitat de l'Aprenentatge**: L'estabilitat de l'aprenentatge podria ser més baixa en l'escenari dels escacs a causa de la complexitat de les transicions d'estats i la necessitat de generalitzar a través d'una varietat més àmplia de situacions.
- **Generalització i Sobreajustament**: La *Q Table* del problema de la graella pot representar fàcilment tots els estats i accions, reduint el risc de sobreajustament. En contrast, la *Q Table* dels escacs ha de generalitzar a partir de menys experiències a través d'un espai d'estats molt més gran, augmentant el risc de sobreajustament als estats i accions observats.
- **Recursos Computacionals**: L'escenari dels escacs requereix més potència computacional i memòria per emmagatzemar i actualitzar la *Q Table*.

En resum, mentre que tots dos escenaris utilitzen *Q-learning* per trobar camins o estratègies òptimes, el problema de la graella és una aplicació més directa amb un aprenentatge més ràpid i més estable, mentre que el problema dels escacs presenta un repte més complex i amb una major demanda computacional amb un procés d'aprenentatge que és més sensible a les especificitats de la implementació, com ara l'estratègia d'exploració i la configuració de la recompensa.

(e) **Compare the use of Q-learning with that of search algorithms of P1 for the chess scenario seen here, both in the deterministic and stochastic case.**
En el context de l'escenari d'escacs presentat en les dades proporcionades, comparar l'ús de Q-learning amb els algorismes de cerca (Cerca en Profunditat, Cerca en Amplada i Cerca A* amb una heurística personalitzada) implica considerar diversos factors clau com ara estratègies d'exploració, component d'aprenentatge, adaptabilitat a entorns canviants i complexitat computacional. A continuació fem un resum per cadascun dels casos:

Cas Determinista:

a) **Completitud i optimització:**
- **Algorismes de Cerca:** Estan dissenyats per trobar una solució si n'hi ha (completitud) i es poden adaptar per trobar la solució òptima (optimització), com ara A* amb una bona heurística.
- **Q-learning:** És un algorisme d'aprenentatge per reforç que pot aprendre la política òptima, però pot necessitar de moltes iteracions per convergir, especialment en un entorn determinista on l'exploració pot ser limitada.

b) **Rendiment:**
- **Algorismes de Cerca:** En un entorn determinista, el rendiment dels algorismes de cerca és generalment previsible. Per exemple, la Cerca A* actuarà de manera consistent basant-se en la seva heurística.
- **Q-learning:** Pot trigar més temps a aprendre els moviments òptims, però un cop après, pot actuar molt bé i bastant ràpid.

c) **Recursos Computacionals:**
- **Algorismes de Cerca:** Poden ser computacionalment costosos, amb BFS i DFS explorant potencialment un gran nombre de nodes. A* pot ser més eficient amb una bona heurística.
- **Q-learning:** Requereix memòria per emmagatzemar la taula Q i potència computacional pel procés d'aprenentatge, que pot ser considerable depenent de la grandària de l'espai d'estats i accions.

Cas Estocàstic:

a) **Robustesa a Canvis:**
- **Algorismes de Cerca:** Generalment no tenen en compte entorns estocàstics i poden no actuar bé sense modificacions.
- **Q-learning:** És més robust davant entorns estocàstics ja que aprèn de l'experiència i actualitza la seva política basada en els resultats observats.

b) **Adaptabilitat:**
- **Algorismes de Cerca:** Aquests algorismes no s'adapten en temps real; cal tornar a executar-los per respondre a canvis.
- **Q-learning:** Continuament actualitza els seus valors Q basant-se en recompenses, la qual cosa permet adaptar-se a canvis en l'entorn.

c) **Exploració vs. Explotació:**
- **Algorismes de Cerca:** Són purament explotatius, utilitzant la informació proporcionada per prendre decisions.
- **Q-learning:** Equilibra l'exploració i l'explotació, que és crític en entorns estocàstics per descobrir les millors estratègies.

En resum, per a escenaris d'escacs deterministes, els algorismes de cerca amb una heurística ben definida poden ser molt eficients i efectius. No obstant això, en escenaris estocàstics, *Q-learning* pot ser preferible a causa de la seva capacitat d'adaptació i aprenentatge de l'entorn. La tria entre aquests mètodes dependrà dels requeriments específics de la tasca, incloent la necessitat d'adaptabilitat, aprenentatge al llarg del temps i recursos computacionals disponibles.

(f) **Use your Q-learning implementation of the drunken sailor on the second board configuration of P1, and try to find the best parameter combination for quick convergence. How robust was your previous parameter choice? Discuss your results**
En la segona configuració del taullel en la P1 només canviava la posició inicial del rei blanc, passava de (7, 4) a (7,7). Per aquest cas, hem escollit els següents paràmetres que ens han donat una convergència bastant ràpida (en uns 1300 episodis):

- α = 0.5 (*learning rate*)
- γ = 0.9 (*discount factor*)
- ϵ = 0.95 (*exploration rate*
- stochasticity_rate = 0.85 (voldrà dir que el 5% de les accions seran aleatòries)

Podem veure que mantenint els mateixos paràmetres que havíem escollit en apartats anteriors, aconseguim una convergència més ràpida, tot i haver augmentat l'aleatorietat amb el *drunken sailor*.

El camí que obtenim és el següent (consisteix en 6 accions per arribar al *check-mate*): [[7, 0, 2], [7, 7, 6]] -> [[0, 0, 2], [7, 7, 6]] -> [[0, 0, 2], [6, 6, 6]] -> [[0, 0, 2], [5, 7, 6]] -> [[0, 0, 2], [4, 6, 6]] -> [[0, 0, 2], [3, 5, 6]] -> [[0, 0, 2], [2, 4, 6]].

4 PROVES REALITZADES

Les proves realitzades per tal de dur a terme tots els exercicis que es demanaven al document de la pràctica les hem deixat comentades i visibles dins del propi codi de *aichess.py* per l'exercici dels escacs, i dins del fitxer *p3_ia_ex1.py* pel primer exercici.

Per cadascun dels exercicis hem creat una funció (e.g. ***def chess_a(self, TA)***) o bé (e.g. ***def section_a(...)***, pel cas del primer exercici) que executa les simulacions demanades.

Per tal d'executar les proves només cal executar el fitxer *aichess.py* i s'imprimiran els diferents resultats. De mateixa manera, per veure les diferents seccions demanades en l'exercici 1, només cal executar el fitxer *p3_ia_ex1.py*.

5 CONCLUSIONS

L'objectiu principal d'aquesta pràctica ha estat familiaritzar-nos amb l'aprenentatge per reforç, específicament amb l'algorisme *Q-learning*. Aquesta experiència ens ha permès comprendre millor com aquests algoritmes poden resoldre problemes complexos de manera semi-supervisada.

Hem aplicat *Q-learning* en diversos contextos: primer en un senzill *grid*, tant amb com sense l'element del mariner ebri, i posteriorment en una configuració d'escacs, repetint l'exercici amb i sense aquesta variable. Aquesta diversitat d'aplicacions ha mostrat la versatilitat del *Q-*

learning en diferents entorns.

La pràctica ha requerit una dedicació considerable, centrada no només en la comprensió dels fonaments teòrics del *Q-learning* sinó també en la seva aplicació pràctica. Aquest enfocament ens ha permès veure com la teoria es tradueix en solucions pràctiques.

Els resultats obtinguts han estat molt positius, mostrant que és possible aconseguir una comprensió sòlida del *Q-learning* i la seva aplicació en problemes variats. La capacitat de l'algorisme per adaptar-se a diferents escenaris ha estat particularment impressionant.

En conclusió, aquesta pràctica ha estat una experiència enriquidora i educativa, brindant-nos una comprensió profunda de l'algorisme *Q-learning* i la seva aplicabilitat en l'aprenentatge per reforç.