Universitat de Barcelona

Report Artificial Intelligence

Practical III - Reinforcement Learning

Arturo Latorre
Muhammad Mohsin Zaman

Introducción

En esta tercera práctica, primeramente se nos ha proporcionado un tablero 3x4, donde hay un estado bloqueado en la posición [1,1], y la posición inicial en el [0,0], y la posición final [2, 3]. Para llevar a cabo esta primera parte, hay que tener en cuenta las diferentes recompensas que se nos plantea, y el problema del Drunken Sailor, y aplicar el algoritmo Q-Learning para resolver el problema.

Para la segunda parte de la practica, tenemos que volver a la configuración de la primera practica, donde se nos ha proporcionado una configuración inicial específica del tablero que contiene un rey negro en la posición [0,4] y dos piezas blancas: un rey en la posición [7,4] y una torre en la posición [7,0].

Para llevar a cabo esta práctica, debemos tener en cuenta que las piezas negras no se mueven, y solo yo puedo mover las piezas blancas. El objetivo principal es encontrar una configuración de estado que resulte en un jaque mate para el rey negro.

Para lograrlo, se nos pide encontrar este estado mediante la implementación de un algoritmo básico implementado con el algoritmo Q-Learning.

Preguntas

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.

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

Las acciones van dependiendo de los valores que pongas en epsilon, para hacer el epsilon-greedy, la alpha, y la gamma. Actualmente hemos empezado con epsilon=0.5, gamma=0.9, alpha=0.1.

```
(Abajo) --> (2, 0)   Reward: -1
+---------------+
| - | - | - | - |
| - | X | - | - |
| 0 | - | - | - |
+---------------+

(Arriba) --> (1, 0)   Reward: -1
+---------------+
| - | - | - | - |
| 0 | X | - | - |
| - | - | - | - |
+---------------+

(Abajo) --> (2, 0)   Reward: -1
+---------------+
| - | - | - | - |
| - | X | - | - |
| 0 | - | - | - |
+---------------+

(Arriba) --> (1, 0)   Reward: -1
+---------------+
| - | - | - | - |
| 0 | X | - | - |
| - | - | - | - |
+---------------+

(Arriba) --> (0, 0)   Reward: -1
+---------------+
| 0 | - | - | - |
| - | X | - | - |
| - | - | - | - |
+---------------+
```

```
(Derecha) --> (0, 1)   Reward: -1
+---------------+
| - | 0 | - | - |
| - | X | - | - |
| - | - | - | - |
+---------------+

(Izquierda) --> (0, 0)   Reward: -1
+---------------+
| 0 | - | - | - |
| - | X | - | - |
| - | - | - | - |
+---------------+

(Derecha) --> (0, 1)   Reward: -1
+---------------+
| - | 0 | - | - |
| - | X | - | - |
| - | - | - | - |
+---------------+

(Arriba) --> (0, 1)   Reward: -1
+---------------+
| - | 0 | - | - |
| - | X | - | - |
| - | - | - | - |
+---------------+

(Izquierda) --> (0, 0)   Reward: -1
+---------------+
| 0 | - | - | - |
| - | X | - | - |
| - | - | - | - |
+---------------+
```

```
(Derecha) --> (0, 1)   Reward: -1
+---------------+
| - | 0 | - | - |
| - | X | - | - |
| - | - | - | - |
+---------------+

(Derecha) --> (0, 2)   Reward: -1
+---------------+
| - | - | 0 | - |
| - | X | - | - |
| - | - | - | - |
+---------------+

(Derecha) --> (0, 3)   Reward: 100
+---------------+
| - | - | - | 0 |
| - | X | - | - |
| - | - | - | - |
+---------------+
```

Hay que aclarar que la secuencia de acciones no siempre sera la misma, ya que hay una exploracion aleatoria provocada por el epsilon-greedy. Es por eso que hemos añadido una sola secuencia de acciones.

- **After trying for a bit, what is your parameter choice for alpha, gamma and epsilon? Why?**

Hemos realizado pruebas diferentes con diferentes valores cada una:

- Alpha=0.1, Epsilon=0.5, Gamma=0.9:
  Con estos parámetros, al tener una tasa baja de aprendizaje, es decir, aprende lentamente, y tener una aleatoriedad alta, hace que converja con muchos episodios, hemos runeado un par de veces y podemos observar que converge entre 550 y 850 episodios, cosa que es normal con estos parámetros, provocando una diferencia muy grande
- Alpha=0.3, Epsilon=0.5, Gamma=0.9:
  En este caso, al subir la tasa de aprendizaje, hace que la diferencia entre episodios sea mucho menor, al runearlo un par de veces hemos observado que va entre 220 y 290 episodios.
- Alpha=0.3, Epsilon=0.1, Gamma=0.9:
  Al bajar la tasa de aleatoriedad, y tener una tasa de aprendizaje alto, hace que tengas menos episodios, pero no nos interesa que la tasa de aleatoriedad sea tan baja ya que queremos que haya mas aleatoriedad, aunque eso provoque que hayan menos episodios hasta la convergencia.
- Alpha=0.2, Epsilon=0.5, Gamma=0.9:
  En este caso, donde hay una tasa de aleatoriedad alta, y una tasa de aprendizaje normal, encontramos una diferencia algo alta, pero no hay mucha estabilidad ya que esta diferencia sigue siendo alta, entre 320 episodios a 450 episodios.

En conclusión, los parámetros elegidos son Alpha=0.3, Epsilon=0.5, Gamma=0.9, ya que tenemos una tasa de aprendizaje un poco mas alta que provoca que no haya tanta diferencia entre una ejecución y otra, y además, tenemos la aleatoriedad alta, provocando que cada ejecución, sea muy diferente.

- **How do you judge convergence of the algorithm? How long does it take to converge?**

  Para la convergencia del algoritmo comparamos las dos qtables seguidas, la anterior y la actual, hacemos la diferencia en valor absoluto, y cogemos el valor máximo, si este valor máximo es menos al umbral que hemos seleccionado (0.0001), cuenta la convergencia. Además, hacemos que no sea únicamente una vez, sino que tiene que ser varias veces seguidas, sino esta convergencia se reinicia.

```
# Verificar convergencia
if episodio > 0 and np.max(np.abs(self.Q - self.Q_anterior)) < umbral_convergencia:
    convergencias += 1
    if convergencias >= num_convergencias_necesarias:
        print(f"Convergencia alcanzada en el episodio {episodio}. Imprimiendo acciones:\n")
        for accion, current_coords, next_coords, reward in acciones:
            self.print_action(accion, current_coords, next_coords, reward)
        break
else:
    convergencias = 0
```

Depende de los valores de alpha, gamma y epsilon, pero si lo hacemos con los valores anteriormente comentados, tarda en alrededor de 220 y 290 episodios para la convergencia.

b) Try implementing the more accurate reward given

Answer the questions of the previous section for this case:

What sequence of actions do you obtain? What is your parameter choice for alpha, gamma and epsilon and why? How do you judge convergence of the algorithm? How long does it take to converge?

Como se ha dicho en la respuesta anterior, la secuencia de acciones nunca va a ser la misma, ya que hay un factor de aleatoriedad, pero aqui añadimos una secuencia de acciones con los parametros que hemos seleccionado en la respuesta anterior (alpha=0.3, epsilon=0.5, gamma=0.9):

```
(Derecha) --> (2, 1)   Reward: -4
+---------------+
| - | - | - | - |
| - | X | - | - |
| - | 0 | - | - |
+---------------+

(Derecha) --> (2, 2)   Reward: -3
+---------------+
| - | - | - | - |
| - | X | - | - |
| - | - | 0 | - |
+---------------+

(Abajo) --> (2, 2)   Reward: -3
+---------------+
| - | - | - | - |
| - | X | - | - |
| - | - | 0 | - |
+---------------+

(Izquierda) --> (2, 1)   Reward: -4
+---------------+
| - | - | - | - |
| - | X | - | - |
| - | 0 | - | - |
+---------------+

(Izquierda) --> (2, 0)   Reward: -5
+---------------+
| - | - | - | - |
| - | X | - | - |
| 0 | - | - | - |
+---------------+
```

```
(Derecha) --> (2, 1)   Reward: -4
+---------------+
| - | - | - | - |
| - | X | - | - |
| - | 0 | - | - |
+---------------+

(Derecha) --> (2, 2)   Reward: -3
+---------------+
| - | - | - | - |
| - | X | - | - |
| - | - | 0 | - |
+---------------+

(Arriba) --> (1, 2)   Reward: -2
+---------------+
| - | - | - | - |
| - | X | 0 | - |
| - | - | - | - |
+---------------+

(Arriba) --> (0, 2)   Reward: -1
+---------------+
| - | - | 0 | - |
| - | X | - | - |
| - | - | - | - |
+---------------+

(Derecha) --> (0, 3)   Reward: 100
+---------------+
| - | - | - | 0 |
| - | X | - | - |
| - | - | - | - |
+---------------+
```

Para la elección de parámetros, vamos a probar los mismos parámetros que hemos escogido para realizar las pruebas anteriores:

- Alpha=0.1, Epsilon=0.5, Gamma=0.9:
  Como hemos comentado con las recompensas anteriores, al tener una tasa de aleatoriedad alta, y una tasa de aprendizaje baja, provoca que haya mucha diferencia para llegar a la convergencia entre una ejecución y otra, en este caso con recompensas diferentes va entre 620 episodios y 820 episodios.
- Alpha=0.3, Epsilon=0.5, Gamma=0.9:
  Nos provoca el mismo resultado que la prueba anterior, provoca que la diferencia entre episodios sea mucho menor, en este caso va de 215 episodios a 280 episodios, un resultado un poco menor al anterior, cosa que es normal por el sistema de recompensas.
- Alpha=0.3, Epsilon=0.1, Gamma=0.9:
  Mismo resultado que anteriormente, provoca que hayan menos episodios porque hay una tasa alta de aprendizaje y una aleatoriedad baja, pero no nos interesa que la aleatoriedad sea baja, necesitamos que haya una exploración aleatoria mas alta.
- Alpha=0.2, Epsilon=0.5, Gamma=0.9:
  Resultado parecido al anterior, donde va de 315 episodios a 440 episodios, no nos da tanta estabilidad como nos puede dar con el alpha=0.3, pero podría ser una opción a tener en cuenta. Si tuviésemos que escoger dos diferentes, esta seria la segunda mejor opción.

En conclusión, los parámetros elegidos son Alpha=0.3, Epsilon=0.5, Gamma=0.9, como hemos comentado anteriormente, nos interesa que no haya tanta diferencia entre ejecuciones, y a la vez, nos haga diferentes caminos.

En caso de la convergencia, la evaluamos de la misma manera que antes, solo que ahora las recompensas son diferentes, comparamos los valores de las q-table anterior y la qtable actual en valor absoluto, y cogemos el valor máximo y se compara con el umbral de convergencia (0.0001). Y además, añadimos que tengan que ser 5 veces seguidas para que converja, no solamente con 1 vez. Y con los parámetros seleccionados, nos da unos episodios entre 220 y 300 episodios para converger.

- **What is the effect of the new reward function on performance?**

```
(Arriba) --> (1, 0)   Reward: -4
+---------------+
| - | - | - | - |
| 0 | X | - | - |
| - | - | - | - |
+---------------+

(Arriba) --> (0, 0)   Reward: -3
+---------------+
| 0 | - | - | - |
| - | X | - | - |
| - | - | - | - |
+---------------+

(Derecha) --> (0, 1)   Reward: -2
+---------------+
| - | 0 | - | - |
| - | X | - | - |
| - | - | - | - |
+---------------+

(Derecha) --> (0, 2)   Reward: -1
+---------------+
| - | - | 0 | - |
| - | X | - | - |
| - | - | - | - |
+---------------+

(Derecha) --> (0, 3)   Reward: 100
+---------------+
| - | - | - | 0 |
| - | X | - | - |
| - | - | - | - |
+---------------+
```

El efecto que tiene el nuevo sistema de recompensas en el algoritmo es que deberia ser mas eficiente, ya que cuanto más cerca del "Goal", más recompensa tienes, y por eso, irá más rápido o debería ir más rápido al Goal. Cosa que se puede observar en los rangos de episodios hasta converger. En este caso podemos observar que la pieza intenta moverse lo más rápido al Goal, ya que hay una recompensa mejor si te vas acercando.

- **How does this relate to the search algorithms studied in P1? Could you apply one of those in this case?**

  Es una búsqueda informada al tener este sistema de recompensas, ya que cuanto más cerca del Goal, menor es el castigo, y es por eso, que se puede relacionar con el algoritmo AStar de la P1. Pero A* es un algoritmo determinista y no se puede aplicar a este caso sino hacer modificaciones necesarias para adaptarlo que funcione con escenarios estocásticos lo cual no es algo ideal y práctico.

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.**
Use at least one of the two rewards proposed:
Utilizamos las recompensas diferentes.

1) **What is your parameter choice? Why?**

- Alpha=0.1, Epsilon=0.5, Gamma=0.9:
  Al tener una tasa de aleatoriedad alta, y una tasa de aprendizaje baja, y teniendo el Drunken Sailor, ocurre lo mismo que antes, una diferencia entre episodios muy alta, entre 575 y 845 episodios, cosa que no da estabilidad.
- Alpha=0.3, Epsilon=0.5, Gamma=0.9:
  Con el drunken, la diferencia entre episodios es algo mayor a la anterior ya que va entre 200 episodios y 290 episodios. Es normal ya que tienes una aleatoriedad alta con el epsilon, y además, añades la aleatoriedad del Drunken Sailor.
- Alpha=0.3, Epsilon=0.4, Gamma=0.9:
  Vistos los resultados de la anterior prueba, hemos bajado el epsilon algo más, para ver si dicha diferencia es algo menor. En este caso nos va de 260 episodios a 310 episodios. En el caso del drunken, hay que bajar algo el epsilon, porque hay que contar con dicha aleatoriedad de la embriaguez.
- Alpha=0.2, Epsilon=0.4, Gamma=0.9:
  En este caso nos va de 360 episodios a 555 episodios, cosa que era el resultado esperado, pero no es util, ya que no nos da una estabilidad, y nos da un margen muy alto, cosa que no es interesante para los resultados.

En conclusión, la mejor decisión de parámetros es Alpha=0.3, Epsilon=0.4, Gamma=0.9, ya que nos da una estabilidad menor a la de los demás parámetros, y nos da un equilibrio.

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**
   En el drunken sailor, podemos observar que tarda entre 6 y 13 acciones para llegar al Goal, ya que depende de la aleatoriedad, y en cambio en el caso sin drunken sailor, es decir, con el escenario determinista, tarda entre 6 y 12 acciones para llegar al Goal, porque también depende de la aleatoriedad, pero un poco menos.

3) **What is the optimal path found? If we watched the sailor try to follow it, would they always follow the same path?**

   El path más óptimo, teniendo en cuenta que solo explotamos, es decir, que no dependa de la aleatoriedad del epsilon, siempre es el mismo path, pero hay un 1% que no siempre lo sea, cosa que es muy difícil que suceda.

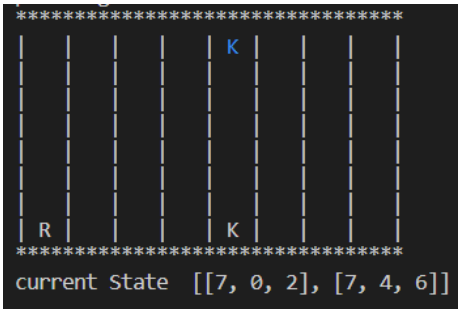
```
(Derecha) --> (2, 1)   Reward: -4
+---------------+
| - | - | - | - |
| - | X | - | - |
| - | O | - | - |
+---------------+

(Derecha) --> (2, 2)   Reward: -3
+---------------+
| - | - | - | - |
| - | X | - | - |
| - | - | O | - |
+---------------+

(Arriba) --> (1, 2)   Reward: -2
+---------------+
| - | - | - | - |
| - | X | O | - |
| - | - | - | - |
+---------------+

(Derecha) --> (1, 3)   Reward: -1
+---------------+
| - | - | - | - |
| - | X | - | O |
| - | - | - | - |
+---------------+

(Arriba) --> (0, 3)   Reward: 100
+---------------+
| - | - | - | O |
| - | X | - | - |
| - | - | - | - |
+---------------+
```

4) **Could we apply one of the algorithms in P1 here? Why?**

   En la P1, el algoritmo que implementamos es el A* que como sabemos, es ideal y funciona muy bien cuando se trata de escenarios deterministas. En esta práctica como estamos trabajando con Q-learning que trata más escenarios estocásticos (probabilidades) no podríamos aplicar A* tal cual sin modificaciones necesarias para trabajar con escenarios estocásticos.

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.**

Para este ejercicio, utilizamos la siguiente configuración del tablero



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

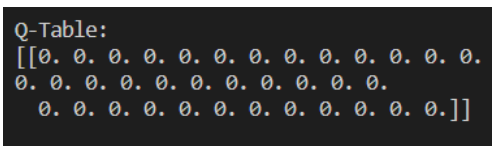
Como sabemos, el número de episodios no es siempre exacto por lo cual, ejecutamos el programa dos veces para hacernos la idea de cuántos episodios se necesitan hasta que converja utilizando los parámetros siguientes:

- Alpha=0.1, Epsilon=0.1, Gamma=0.9:

Hemos notado que aproximadamente necesita 600-800 episodios por lo tanto, a continuación adjuntamos q-table inicial, los dos intermedios en el episodio 300 y 600 y el final.

Nota: Debido a que los q-tables son bastantes grandes en dimensión, aquí adjuntamos estos en forma de numpy array al cual Python lo muestra en forma reducida. No obstante, en el siguiente apartado, los q-tables(inicial, intermedios y final) correspondientes a los mejores paràmetros(alpha,epsilon,gamma) los adjuntamos en ficheros ".txt" junto a la entrega para poder visualizar q-table enteros en formato original "nested dictionary".

Q-table inicial:



Q-table intermedios:

Q-table episode 300:

```
Q-TABLE AT EPISODE  300

Q-Table:
[[ 0.     -0.1     0.     ...  0.     0.     0.    ]
 [ 0.     -0.1     0.     ...  0.     0.     0.    ]
 [ 0.      0.      0.     ...  0.     0.     0.    ]
 ...
 [-0.19    0.     -0.3439 ...  0.     0.     0.    ]
 [-0.1     0.      0.     ...  0.     0.     0.    ]
 [-0.1     0.      0.     ...  0.     0.     0.    ]]
```

Q-table episode 600:

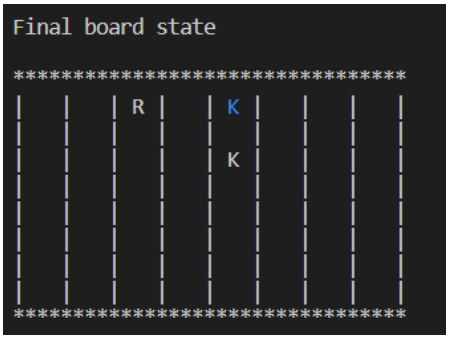
```
Q-TABLE AT EPISODE  600

Q-Table:
[[ 0.     -0.1     0.      ...  0.     0.     0.    ]
 [ 0.     -0.1     0.      ...  0.     0.     0.    ]
 [ 0.      0.      0.      ...  0.     0.     0.    ]
 ...
 [-0.271   0.     -0.40951 ...  0.     0.     0.    ]
 [-0.19    0.      0.      ...  0.     0.     0.    ]
 [-0.1     0.     -0.1     ...  0.     0.     0.    ]]
```

Q-table final:

```
Q-table final:

Q-Table:
[[ 0.      -0.1      0.       ...  0.       0.       0.     ]
 [ 0.      -0.1      0.       ...  0.       0.       0.     ]
 [ 0.       0.       0.       ...  0.       0.       0.     ]
 ...
 [-0.271    0.      -0.40951  ...  0.       0.       0.     ]
 [-0.19     0.       0.       ...  0.       0.       0.     ]
 [-0.1      0.      -0.1      ...  0.       0.       0.     ]]
```

La secuencia de acciones y el estado final del tablero que obtenemos el la siguiente:

```
Best path to checkmate is:
[[[7, 0, 2], [7, 4, 6]], [[0, 0, 2], [7, 4, 6]], [[0, 0, 2], [6, 3, 6]], [[0, 2, 2], [6, 3, 6]], [[0, 2, 2], [5, 3, 6]], [[0, 2, 2], [4, 4, 6]], [[0,
 2, 2], [3, 4, 6]], [[0, 2, 2], [2, 4, 6]]]
```



ii) After trying for a bit, what is your parameter choice for alpha, gamma and epsilon? Why?

Para responder a esta pregunta, hemos realizado 3 pruebas con diferentes parámetros a fin de averiguar cuál es la mejor combinación.

1. La primera prueba ha sido con los siguientes parámetros:
   - Alpha=0.1, Epsilon=0.4, Gamma=0.9:

   Con estos parámetros, hemos ejecutando 4 veces el programa y como resultado vemos el el aprendizaje no es muy estable ya que el número de episodios que necesita para converger ronda entre 500-950, número que es normal para estos parámetros debido a que hay una tasa alta de aleatoriedad mientras que la tasa de aprendizaje no es suficiente.
2. La segunda prueba es con los siguientes parámetros:
   - alpha=0.3, Epsilon=0.4, Gamma=0.9:

   Por tal de tener más estabilidad, ahora probamos de incrementar la tasa de aprendizaje y mantenemos la misma aleatoriedad con lo cual, los resultados que obtenemos son que en las 3 ejecuciones realizadas, la diferencia entre el número de episodios es relativamente poca donde este va entre 463- 495
3. La tercera y última prueba es con los siguientes parámetros:
   - alpha=0.5, Epsilon=0.3, Gamma=0.9

   Esta prueba por el hecho de tener una tasa de aprendizaje alta junto a una tasa de aleatoriedad suficiente para mantener el grado de aleatoriedad, proporciona resultados similares a la prueba anterior pero con la diferencia de que los episodios que tarda en converger son aún menos (250-380) y el aprendizaje sigue siendo muy estable.

Finalmente, tras haber realizado las pruebas anteriores, llegamos a la conclusión que los mejores parámetros son alpha=0.5, Epsilon=0.3, Gamma=0.9. No obstante, cabe destacar que esto no significa que sean los únicos mejores parámetros ya que pueden haber otros casos pero estos muestran bastantes buenos resultados en las pruebas realizadas.

Como comentamos en la nota del apartado anterior, en la entrega añadiremos los q-tables(inicial, 2 intermedios y final) pertenecientes a la ejecución de los mejores parámetros escogidos(alpha=0.5, Epsilon=0.3, Gamma=0.9) tras estas pruebas cada uno en un fichero ".txt" a fin de poder visualizar los q-values.

iii) How do you judge convergence of the algorithm? How long does it take to converge?

De forma similar al ejercicio 1 de la práctica, para la convergencia del algoritmo comparamos las dos qtables seguidas, la anterior y la actual, hacemos la diferencia en valor absoluto, y cogemos el valor máximo, si este valor máximo es menos al umbral que hemos seleccionado (0.0001), cuenta la convergencia. Además, hacemos que no sea únicamente una sola vez, sino que tiene que ser varias veces seguidas(5 veces) para finalmente poder considerar que ha habido convergencia, sino esta convergencia se reinicia.

```
if episodio > 0:
    if self.has_converged(self.Q, self.Q_anterior, umbral_convergencia):
        convergencias += 1
        if convergencias >= num_convergencias_necesarias:
            self.total_episodes = episodio
            break
else:
    convergencias = 0
```

El hecho de cuanto tarda en converger depende de los parámetros alpha, gamma, epsilon pero si consideramos los parámetros escogidos como los mejores en el apartado (ii), tarda entre 345-420 episodios.

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.

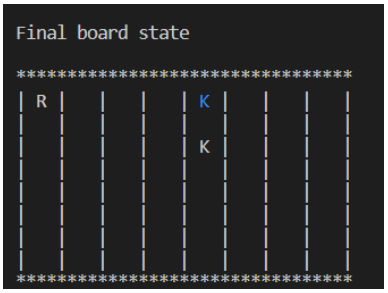
Las preguntas en las sección anterior eran las siguientes:

- **What sequence of actions do you obtain? After trying for a bit, what is your parameter choice for alpha, gamma and epsilon? Why? How do you judge convergence of the algorithm? How long does it take to converge?**

En cuanto a la primera pregunta, Igual que en la sección anterior, utilizamos los mismos parámetros(Alpha=0.1, Epsilon=0.1, Gamma=0.9) pero utilizando recompensas sensibles, la secuencia de acciones que obtenemos es la siguiente:

```
Convergence achieved in 854 episodes.

(Arriba Tower) --> [[0, 0, 2], [7, 4, 6]]   Reward: -5
(Arriba King) --> [[0, 0, 2], [6, 4, 6]]   Reward: -4
(Diagonal Arriba Izq King) --> [[0, 0, 2], [5, 3, 6]]   Reward: -3
(Arriba King) --> [[0, 0, 2], [4, 3, 6]]   Reward: -2
(Diagonal Arriba Der King) --> [[0, 0, 2], [3, 4, 6]]   Reward: -1
(Arriba King) --> [[0, 0, 2], [2, 4, 6]]   Reward: 100

Best path to checkmate is:
[[[7, 0, 2], [7, 4, 6]], [[0, 0, 2], [7, 4, 6]], [[0, 0, 2], [6, 4, 6]], [[0, 0, 2], [5, 3, 6]], [[0, 0, 2], [4, 3, 6]], [[0, 0, 2], [3, 4, 6]], [[0,
 0, 2], [2, 4, 6]]]
```

Y es estado final del tablero es el siguiente:



Para responder la segunda pregunta, de forma similar que en la sección anterior realizaremos tres pruebas utilizando los mismos valores de parámetros para ver la diferencia entre utilizar una recompensa normal y una recompensa más sensible.

1. La primera prueba con los siguientes parámetros:
    - alpha=0.1, Epsilon=0.4, Gamma=0.9:

    Con estos parámetros, donde la tasa de aprendizaje es baja y el nivel de aleatoriedad es mucho más alto, hemos ejecutando el código 3 veces y el número de episodios que ha tardado hasta llegar a convergencia es 744, 994, 1035 algo que era evidente debido a que el agente aprende lentamente y por esa razón tarda más.
2. La segunda prueba con los siguientes parámetros:
    - alpha=0.3, Epsilon=0.4, Gamma=0.9:

    En este caso, también hemos ejecutado 3 veces y se ha llegado a la convergencia en 343, 567 y 529 con lo cual, vemos que hay bastante más estabilidad comparando con la prueba anterior..
3. La tercera y última prueba con los siguientes parámetros:
    - alpha=0.5, Epsilon=0.3, Gamma=0.9

    Tras ejecutar el programa tres veces, donde la tasa de aprendizaje es alta mientras que la tasa de aleatoriedad también está pensada para mantener la aleatoriedad, vemos que el algoritmo converge en 422, 332, 365 episodios.

En conclusión. Tras realizar las previas pruebas con diferentes parámetros, podemos decir que los parámetros (alpha=0.5, Epsilon=0.3, Gamma=0.9) son relativamente buenos para conseguir convergencia en menos episodios manteniendo la aleatoriedad.

ii. What is the effect of the new reward function on performance?

Tras realizar varias pruebas, cabe destacar que el hecho de utilizar la función de recompensas sensibles no ha afectado mucho a mejorar el rendimiento del algoritmo y viendo el número de episodios necesarios para convergencia en las pruebas realizadas del apartado anterior, vemos que son muy cercanos a los de la sección anterior.

c. Drunken sailor

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.

ii. Use any reward you prefer:

Para esta sección, utilizaremos la función de recompensas normales.

1. What is your parameter choice? Why?

- Los parámetros que vamos a utilizar son(alpha=0.5, Epsilon=0.3, Gamma=0.9). La razón de escoger estos parámetros es porque tenemos el resultado aproximado con parámetros similares ((alpha=0.5, Epsilon=0.3, Gamma=0.9)) que hemos visto en apartado 1 donde se tardaba entre (345-420) episodios y ahora la intención es mirar que diferencia provoca el hecho de ser "Drunken Sailor" usando los mismos parámetros. Tras ejecutar un par de veces, el drunken tarda la primera vez 335 y la segunda vez 429 episodios, lo cual es un resultado esperado ya que el hecho de usar porcentaje de realizar la acción escogida es 99% por lo que no cambia mucho el resultado comparando con el del apartado anterior.

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.

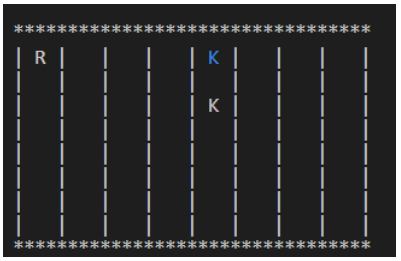
Al igual que pasaba en el escenario 1, la diferencia que hay entre el drunken sailor y con el escenario determinista es muy pequeña, ya que depende del 1%. En el caso del drunken sailor se observa que hace entre 6 y 9 acciones para llegar al checkmate, y en cambio, para el caso determinista, tarda las mismas acciones que el drunken, ya que esta diferencia es tan mínima, que casi no se ve reflejada a la hora de las acciones. Si hablamos de episodios que necesita, es decir, de juegos para satisfacer que ha encontrado la mejor estrategia antes de irse a la cama, en caso del drunken va entre 310 y 425 episodios, y en caso del escenario determinista, va entre 250 y 380 episodios.

3. What is the optimal path found? If we watched the sailor try to follow it, would they always follow the same path?

Teniendo en cuenta que solo explotamos, es decir, que no dependa de la aleatoriedad del epsilon, siempre casi siempre sera el mismo path, pero hay un 1% que hace que tal vez no sea ese path, cosa que es muy difícil que suceda, pero puede suceder.

7 movimientos

```
Best path to checkmate is:
[[[7, 0, 2], [7, 4, 6]], [[0, 0, 2], [7, 4, 6]], [[0, 0, 2], [6, 5, 6]], [[0, 0, 2], [5, 5, 6]], [[0, 0, 2], [4, 5, 6]], [[0, 0, 2], [3, 4, 6]], [[0,
 0, 2], [2, 4, 6]]]
```



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?

Los dos escenarios difieren bastante uno del otro. La diferencia principal es que en el escenario 1, solamente había un agente con un número reducido de acciones. En cambio, en el escenario de Chess, tenemos 2 agentes que son el rey blanco y la torre blanca y cada uno tiene sus movimientos diferentes lo cual hace que este escenario tenga una política de movimientos más extensa que la del escenario 1.

Otra de las diferencias es que en el escenario 1, el estado objetivo era las misma posición siempre mientras que en el escenario de chess, el estado objetivo es cuando llegamos al jaque mate, que pueden ser diferentes posiciones del tablero.

En cuanto al efecto que hace eso en nuestros resultados es que las acciones serán diferentes, ya que como se ha comentado, en el escenario 1, solo hay una pieza, y en el escenario 2, hay dos piezas, y eso provocará que tengas más coordenadas. En cuanto, a los episodios en converger, no hay gran diferencia entre un ejercicio y otro, ya que en el

escenario del chess, buscas unas posiciones exactas, y en cambio, en el escenario 1, la pieza va en busca del goal únicamente.

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.

Comparando el uso de Q-learning con el algoritmo A* de la práctica 1 en caso determinista, sabemos que por un lado, A* al ser un algoritmo de búsqueda es muy efectivo en casos deterministas debido a que las acciones son predecibles y además el entorno es conocido y particularmente en el escenario de chess permite encontrar un camino óptimo de forma rápida. En caso de Q-learning, este también es bastante efectivo en entornos deterministas ya que puede aprender políticas óptimas del entorno y eventualmente llegar al objetivo pero el único punto negativo que puede llegar a tener es que dependiendo de entornos (fáciles o complejos) puede requerir muchas iteraciones para converger.

En el caso estocástico, directamente podemos decir que el A* no es la mejor opción debido a que necesita un entorno conocido y por lo tanto puede requerir de modificaciones varias a fin de poder funcionar pero aun así no llegaría a ser muy óptimo. En cambio, el Q-learning se adapta de manera fácil a escenarios estocásticos ya que puede ir aprendiendo del entorno en tiempo real e ir mejorando hasta llegar a converger.

Conclusiones

En conclusión, hemos logrado la práctica de manera exitosa, y nos ha servido para entender el algoritmo de Q-Learning utilizando en las configuraciones del tablero inicial 4x3, y la configuración que lleva al checkmate en ambos casos.

También, hemos demostrado la diferencia entre las recompensas diferentes en un tablero en caso del primer escenario, y la diferencia entre el drunken sailor y un escenario determinista.