NOMBRE: Bruno Farfán

SECCIÓN: 1

PUNTAJE:

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC3675 — Aprendizaje Reforzado — 1' 2025

Tarea 3 – Respuesta Pregunta a)

Que un algoritmo sea *off-policy* finalmente significa que es capaz de aprender una política recopilando datos del ambiente usando cualquier otra política. Mientras que por otro lado un algoritmo *on-policy* significa que debe usar la misma política óptima que quiere aprender para recopilar la información.

Sarsa es un algoritmo *on-policy* debido a que la actualización de $Q(S, A)$ depende de $Q(S', A')$, y este valor A' es seleccionado dependiendo de la política que se sigue, por lo que la política que se aprende se ve influenciada por la política que se está usando.
Por otro lado, se tiene que **Q-Learning** es *off-policy* ya que en su forma de actualizar los valores $Q(S, A)$ se utiliza la mejor posible acción en el próximo estado $(max_a(QS', a))$, por lo que es independiente de la política para explorar.

NOMBRE: Bruno Farfán

SECCIÓN: 1

PUNTAJE:

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC3675 — Aprendizaje Reforzado — 1' 2025

Tarea 3 – Respuesta Pregunta b)

Considerando el caso en que se tiene un algoritmo **Sarsa** que sigue una política *greedy* se tiene lo siguiente:

1: **Algorithm parameters:** step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
2: Initialize $Q(s, a)$ arbitrarily for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, except that $Q(\text{terminal}, \cdot) = 0$
3: **for** each episode **do**
4: Initialize S
5: Choose A from S using policy derived from Q (greedy)
6: **while** S is not terminal **do**
7: Take action A, observe R, S'
8: Choose A' from S' using policy derived from Q (greedy)
9: $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma Q(S', A') - Q(S, A)\right]$
10: $S \leftarrow S'$
11: $A \leftarrow A'$
12: **end while**
13: **end for**

Mientras que para **Q-Learning** se tiene que:

1: **Algorithm parameters:** step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
2: Initialize $Q(s, a)$ arbitrarily for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, except that $Q(\text{terminal}, \cdot) = 0$
3: **for** each episode **do**
4: Initialize S
5: **while** S is not terminal **do**
6: Choose A from S using policy derived from Q (greedy)
7: Take action A, observe R, S'
8: $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
9: $S \leftarrow S'$
10: **end while**
11: **end for**

Es evidente que la línea 5 de **Sarsa** es igual a la 6 de **Q-Learning**, sin embargo, se también se puede observar que ahora que **Sarsa** es siempre *greedy*, la actualización en 9 es igual a la que hace **Q-Learning** en 8. Es por esto que para valores iniciales iguales, se tendrá que realizarán las mismas acciones y harán las mismas actualizaciones en todo momento. Es decir, **Sarsa** y **Q-Learning** son **equivalentes** para una selección de acciones *greedy*.

NOMBRE: Bruno Farfán

SECCIÓN: 1

PUNTAJE:

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC3675 — Aprendizaje Reforzado — 1' 2025

Tarea 3 – Respuesta Pregunta c)

El resultado de ejecutar el experimento pedido es el siguiente:

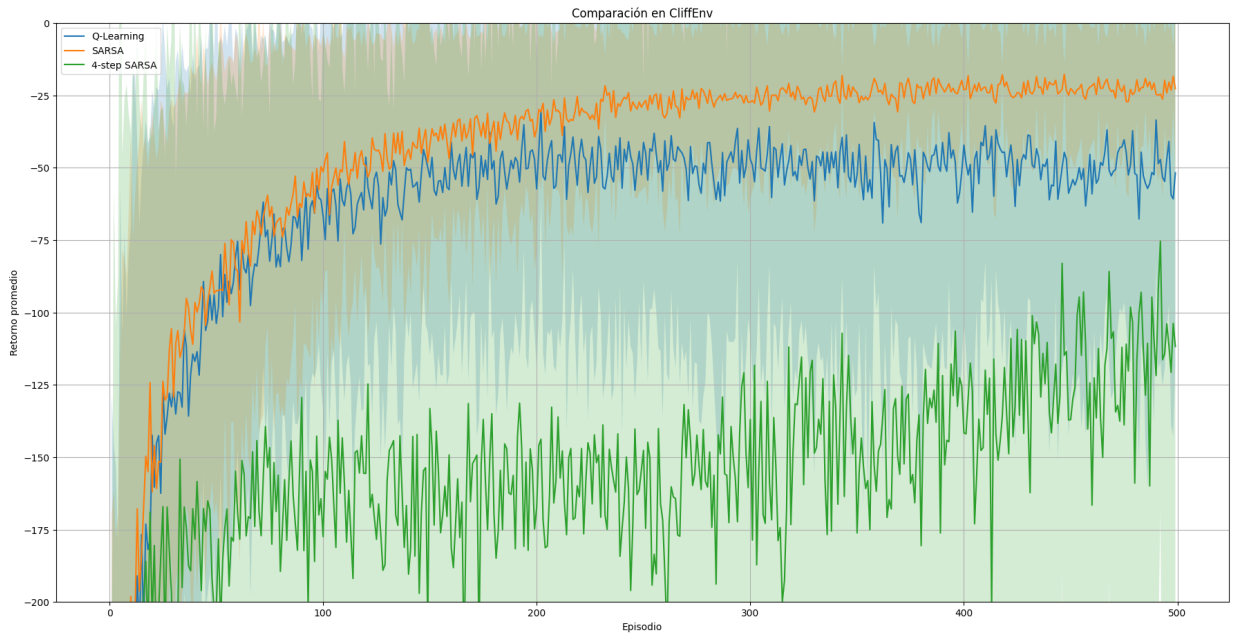

Figure 1: Promedio de 100 ejecuciones en **CliffEnv**, cada una con 500 episodios.

Los resultados obtenidos tienen sentido. Por ejemplo, al comparar **Q-Learning** con **Sarsa**, se ve que el primero es consistentemente peor. Esto se puede explicar porque **Q-Learning** aprende una política óptima pero en el caso *greedy*, es decir, pasar por el camino más corto, pero más riesgoso. Como en el experimento real se tiene que $\epsilon = 0.1$, esta trayectoria tiene una probabilidad no despreciable de caer al *cliff* y perder. Mientras que por otro lado **Sarsa** sí tiene esto en cuenta al momento de aprender, ya que es *on-policy* y por lo tanto tomará un camino sub óptimo, pero seguro, por lo que en promedio tiene una recompensa más alta, al no caer al *cliff* casi nunca.
Esto mismo puede verse también la desviación estándar del gráfico (las sombras de los colores correspondientes), donde **Sarsa** tiene mucho menos varianza que **Q-Learning**.

Por otro lado, comparando **4-step Sarsa** con los otros dos algoritmos, se puede ver que es considerablemente peor que ambos. Esto se debe a un problema similar al que tenía **MonteCarlo**, y es que se actualizan varios valores hacia atrás (4 en este caso), pero como las penalizaciones son muy abruptas y grandes en este ambiente. Esto hace que se sobredimensione el castigo hacia pasos anteriores, y por esto el agente no es capaz

de converger hacia una política más estable y eficiente. Finalmente, el problema es que se está penalizando "rutas" hacia atrás, cuando realmente el castigo debería estar enfocado hacia ese último movimiento que llevó al agente a caer al *cliff* y no a los varios pasos anteriores, los cuales perfectamente podrían ser parte de una trayectoria óptima o relativamente buena.

NOMBRE: Bruno Farfán

SECCIÓN: 1

PUNTAJE:

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC3675 — Aprendizaje Reforzado — 1' 2025

Tarea 3 – Respuesta Pregunta d)

Los resultados del experimento son los siguientes:

Episodio	Dyna - 0	Dyna - 1	Dyna - 10	Dyna - 100	Dyna - 1.000	Dyna - 10.000	RMax
1	-6277.6	-5967.8	-6360.4	-6036.4	-5825.0	-5046.2	-6896.0
2	-3272.6	-2865.0	-2091.6	-2305.2	-1994.6	-2191.0	-281.0
3	-1817.6	-1652.8	-1241.0	-3904.8	-2037.8	-2163.6	-2638.0
4	-1320.0	-2510.2	-2271.8	-3115.6	-1951.0	-766.4	-67.0
5	-1119.6	-1120.4	-1326.4	-478.4	-341.6	-269.4	-67.0
6	-1149.0	-1265.8	-2612.8	-262.8	-123.4	-173.4	-67.0
7	-1295.0	-972.0	-2058.2	-185.8	-276.2	-110.4	-67.0
8	-1360.0	-893.6	-950.0	-387.0	-90.0	-225.2	-67.0
9	-1489.2	-1036.2	-482.0	-133.0	-98.4	-88.0	-67.0
10	-1677.6	-851.0	-640.2	-145.2	-80.4	-94.8	-67.0
11	-1251.6	-653.4	-1274.6	-428.6	-87.4	-83.4	-67.0
12	-1002.2	-667.4	-557.8	-110.4	-113.0	-142.2	-67.0
13	-1067.6	-1129.6	-678.8	-95.2	-87.4	-205.2	-67.0
14	-1732.6	-1337.0	-644.6	-102.0	-97.8	-71.6	-67.0
15	-1839.6	-910.4	-402.0	-88.6	-80.6	-96.0	-67.0
16	-1882.8	-878.0	-745.6	-86.6	-76.8	-102.2	-67.0
17	-1096.8	-770.6	-389.2	-91.8	-87.8	-109.4	-67.0
18	-1352.4	-2005.0	-330.0	-92.8	-78.6	-87.2	-67.0
19	-1245.8	-535.8	-390.6	-169.6	-87.2	-98.8	-67.0
20	-1467.6	-601.2	-1214.4	-90.4	-84.0	-90.0	-67.0

Table 1: Retornos promedio para cada episodio, según el algoritmo utilizado. **Dyna** incluye también sus *planning steps*.

Al observar la tabla transpuesta de retornos, se aprecia que a medida que aumentan los pasos de planificación en Dyna (de 0 a 10.000), los retornos mejoran significativamente. Por ejemplo, en **Dyna - 10.000**, los valores de retorno se acercan a los de **RMax** en varias evaluaciones, indicando que un mayor número de pasos de planificación permite a **Dyna** aproximarse al rendimiento de **RMax**.

Sin embargo, **RMax** muestra un comportamiento distintivo ya que después de una fase inicial de exploración, mantiene retornos constantes y óptimos (-67 en la mayoría de las evaluaciones), lo que sugiere que aprendió una política óptima y la ejecuta consistentemente. En cambio, **Dyna**, incluso con muchos pasos de planificación, sigue mostrando variabilidad en los retornos.
En conclusión, aunque aumentar los pasos de planificación en **Dyna** mejora su rendimiento y lo acerca al de **RMax**, no lo hace equivalente. **RMax** tiene mecanismos de exploración y actualización que le permiten alcanzar y mantener una política óptima, mientras que **Dyna** siempre tendrá variabilidad, por más pasos de planificación que haya.

NOMBRE: Bruno Farfán

SECCIÓN: 1

PUNTAJE:

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC3675 — Aprendizaje Reforzado — 1' 2025

Tarea 3 – Respuesta Pregunta e)

El resultado del experimento realizado es el siguiente:

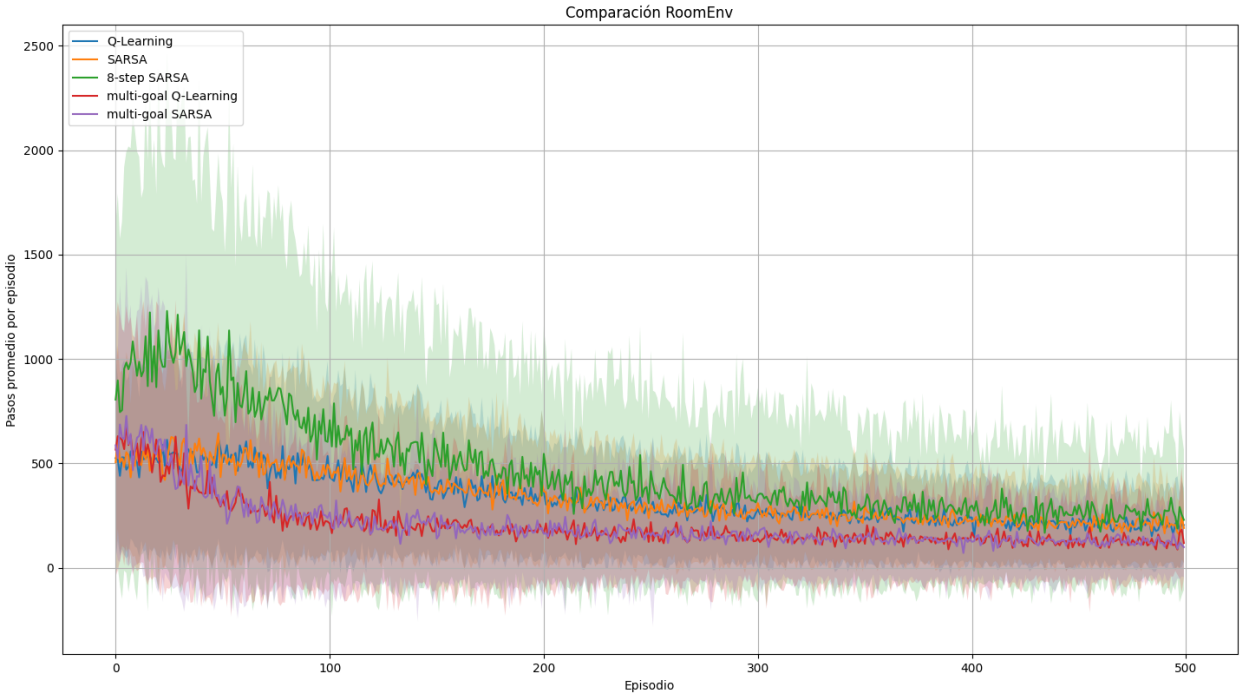

Figure 2: Promedio de 100 ejecuciones en **RoomEnv**, cada una con 500 episodios.

Se puede ver muy claramente que las versiones *multi-goal* tanto de **SARSA** como de **Q-Learning** son mejores que los *baselines*, ya que consitentemente tienen episodos con una menor cantidad de pasos. Esto es consistente con lo planteado en el enunciado, ya que en las versiones *multi-goal* se pueden actualizar las entradas de la tabla para todos los objetivos diferentes al mismo tiempo, mientras que en la versión base es solo uno a la vez.

Por otro lado, también observamos que **multi-goal Q-Learning** y **multi-goal SARSA** tienen rendimientos muy parecidos esto hace sentido, ya que en el ambiente no hay penalizaciones por tomar la ruta más *greedy* siempre (como si había en *Cliff* por ejemplo, donde el riesgo de caer hacía que no fuera la mejor en la práctica, debido al $\epsilon = 0.1$). Es decir, que en la práctica, la ruta *greedy* es la mejor, y por esto la actualización *greedy* u optimista de **Q-Learning** es igual de buena que la actualización realista de **SARSA**.

NOMBRE: Bruno Farfán

SECCIÓN: 1

PUNTAJE:

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC3675 — Aprendizaje Reforzado — 1' 2025

Tarea 3 – Respuesta Pregunta f)

Los resultados de realizar todos los experimentos mencionados en el enunciado se resumen en el siguiente gráfico:

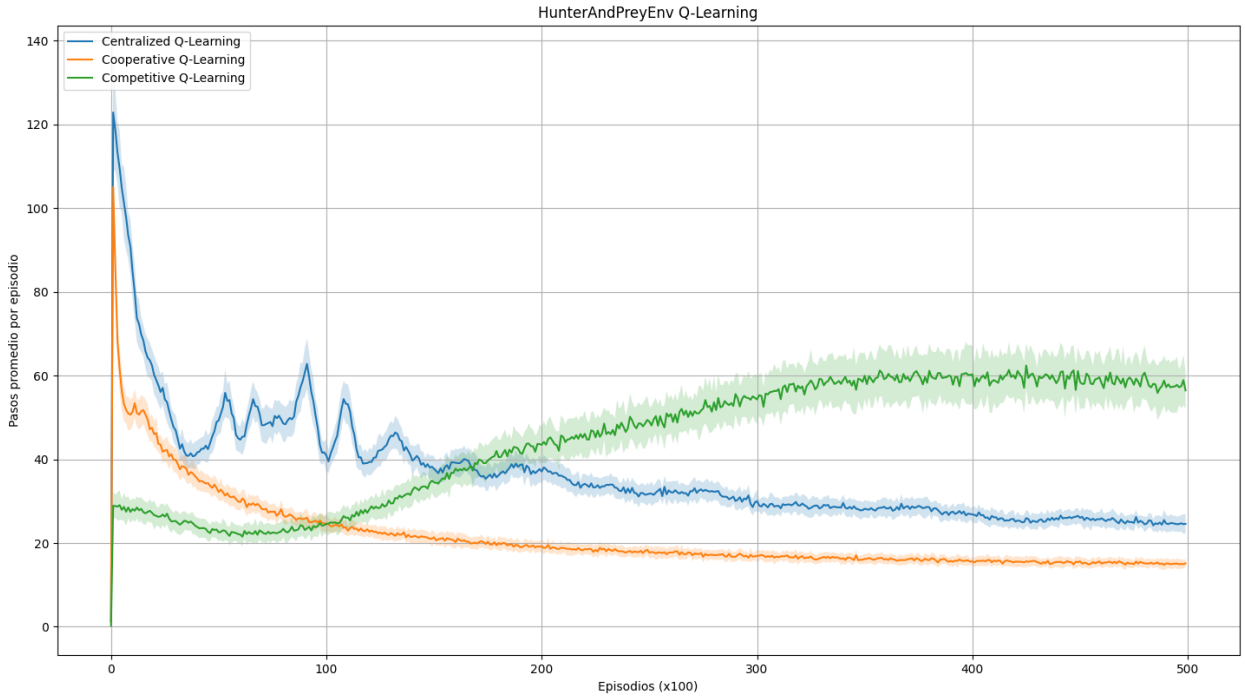

Figure 3: Promedio de 30 ejecuciones en los diferentes ambientes con múltiples agentes, cada una con 50.000 episodios y reportando el valor promerdio cada 100.

La curva azul corresponde a **CentralizedHunterEnv**, la naranja a **HunterEnv** y la verde a **HunterAndPreyEnv**.

Las inestabilidades en la curva azul y su mayor cantidad de pasos en comparación con la naranja, probablemente se deban a que al ser un ambiente centralizado, hay una mucho mayor cantidad de acciones. Debido a esto el aprendizaje es inestable y puede variar rápidamente entre episodios.
Por otro lado, la curva naranja de **HunterEnv** llega mucho más rápido y establemente a un número bajo de pasos por episodio. Se cree que esto se debe a que cada agente tiene un número mucho más pequeño de acciones que realizar y por lo tanto se puede "llenar" más rápido la tabla.

Finalmente, la curva verde de **HunterAndPreyEnv** tiene una bajada inicial, que corresponde al aprendizaje de los dos agentes cazadores, pero luego sube y se estabiliza en alrededor de 60 pasos por episodio. Esto se debe a que el agente **Prey** ahora también aprende, a diferencia de los ambientes anteriores donde tenía una política fija. Esta nueva capacidad de aprendizaje le permite a la presa adaptarse y alargar considerablemente los episodios.
Finalmente se puede observar también que la varianza (sombra de la curva en el gráfico) en la curva verde es mayor, lo que puede deberse a la incapacidad de predecir los movimientos de la presa por parte de los agentes cazadores, mientras que en las otras curvas era más simple al tratarse de una política fija.

NOMBRE: Bruno Farfán

SECCIÓN: 1

PUNTAJE:

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC3675 — Aprendizaje Reforzado — 1' 2025

Tarea 3 – Respuesta Pregunta g)

Los resultados de los experimentos se presentan a continuación:

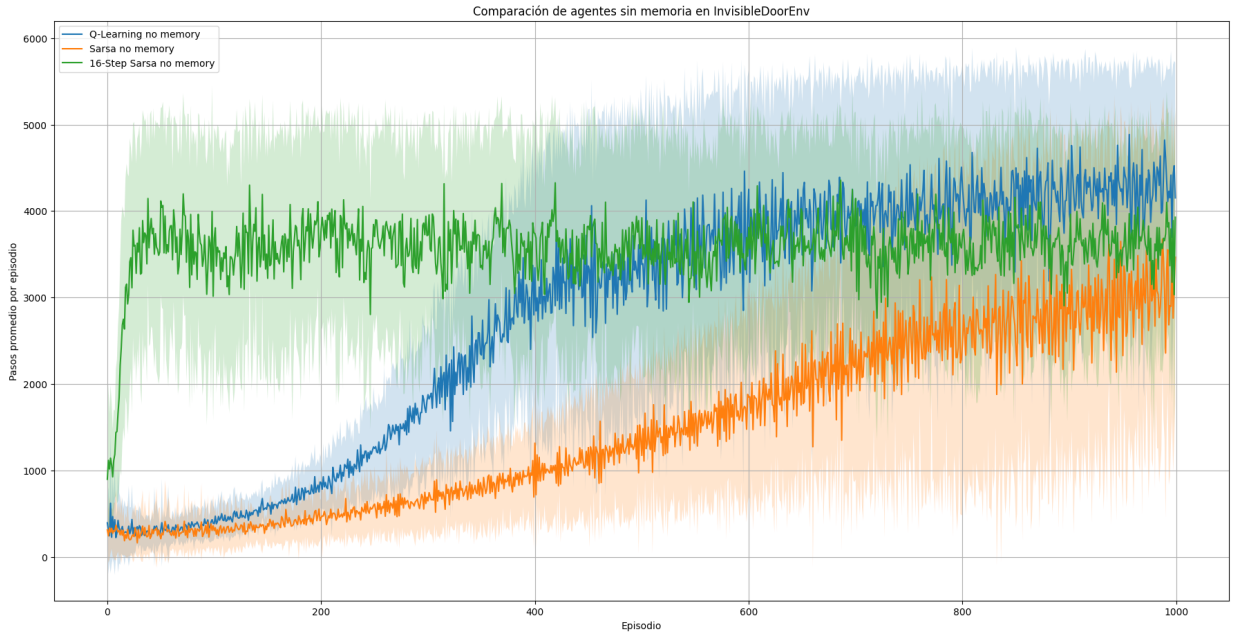

Figure 4: Promedio de 30 ejecuciones con agentes sin memoria en **InvisibleDoorEnv**, cada una con 1000 episodios.

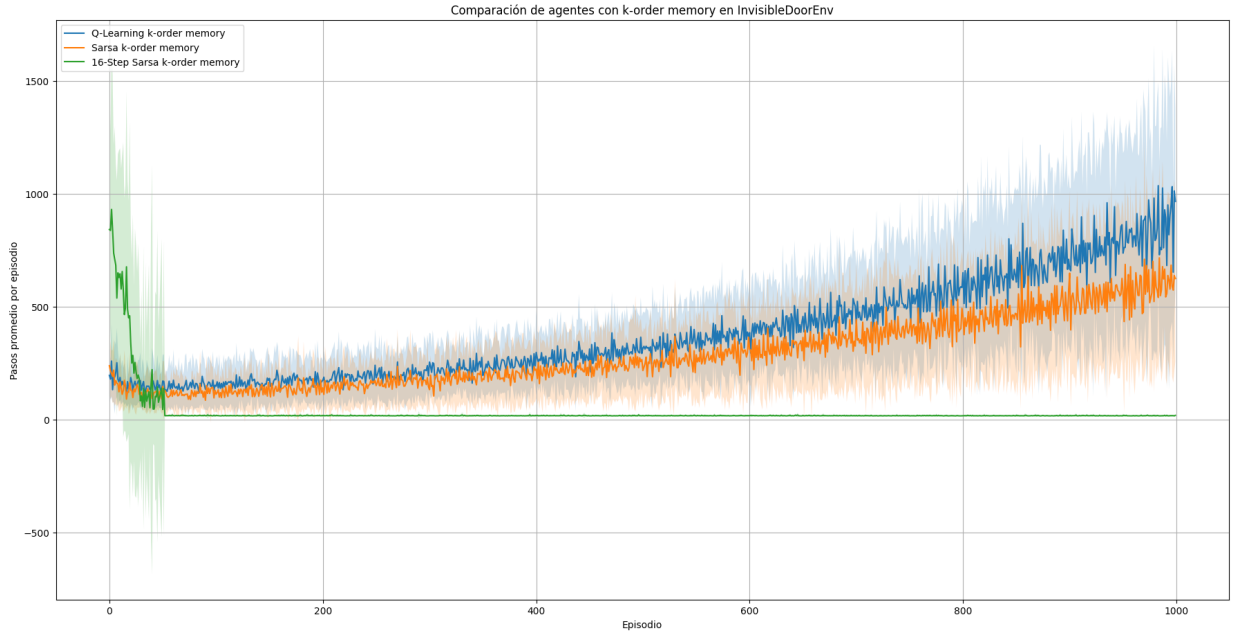

Figure 5: Promedio de 30 ejecuciones con agentes con *k-order memory* en **InvisibleDoorEnv**, cada una con 1000 episodios.

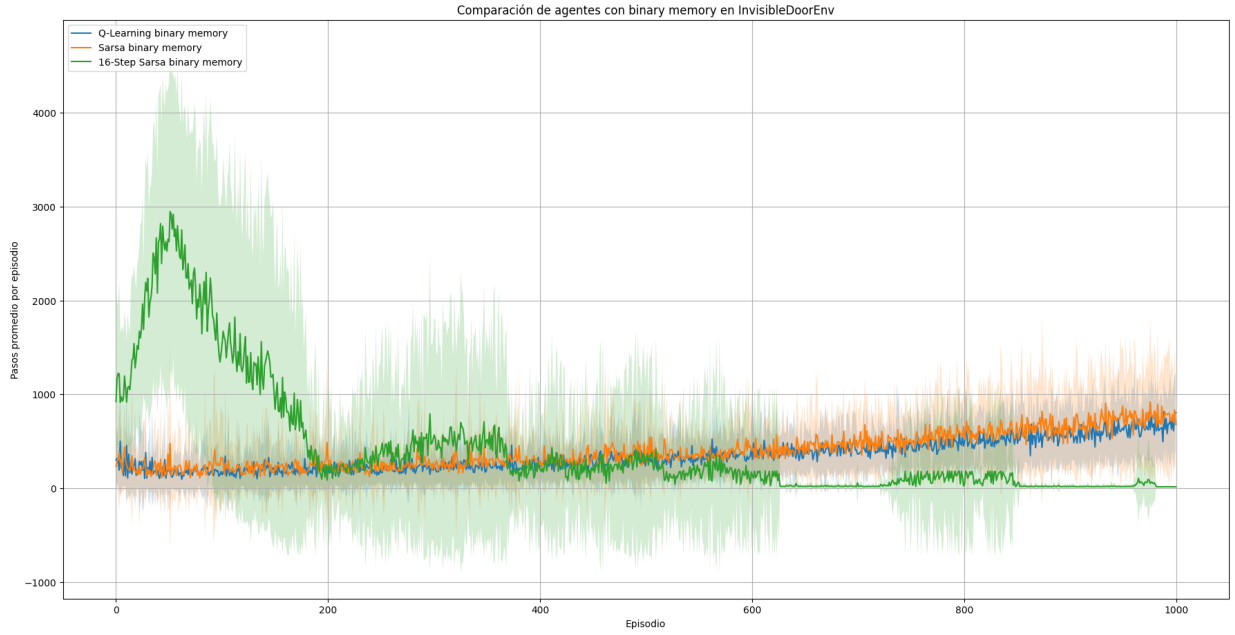

Figure 6: Promedio de 30 ejecuciones con agentes con *binary memory* en **InvisibleDoorEnv**, cada una con 1000 episodios.

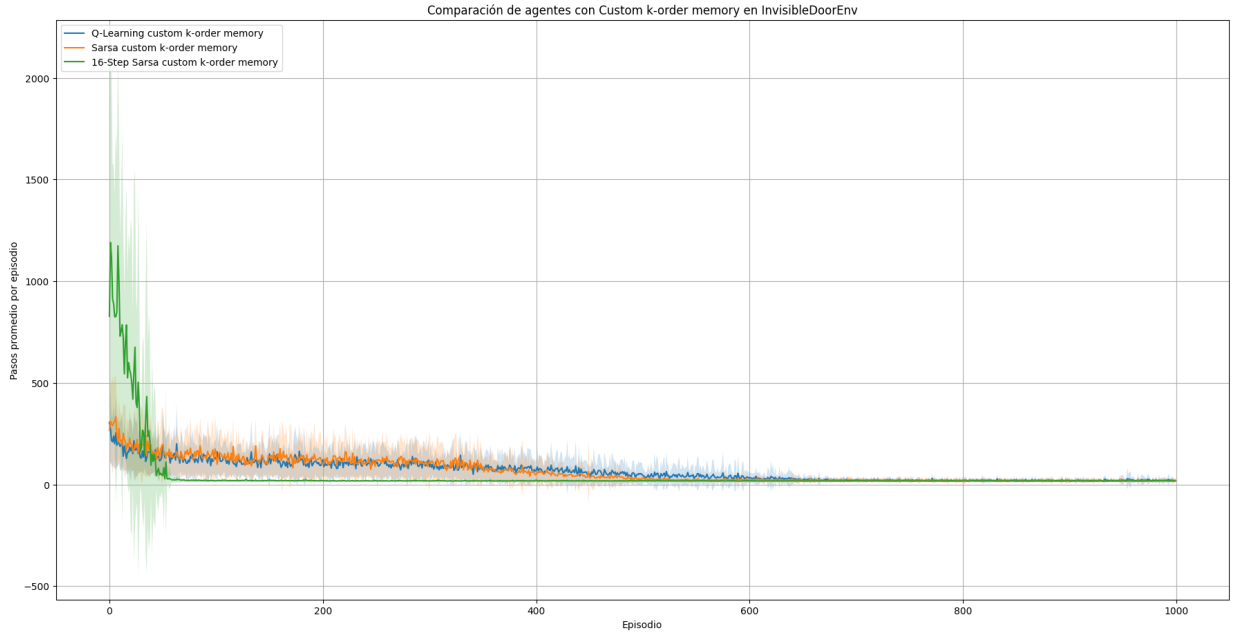

Figure 7: Promedio de 30 ejecuciones con agentes con *custom k-order memory* en **InvisibleDoorEnv**, cada una con 1000 episodios.

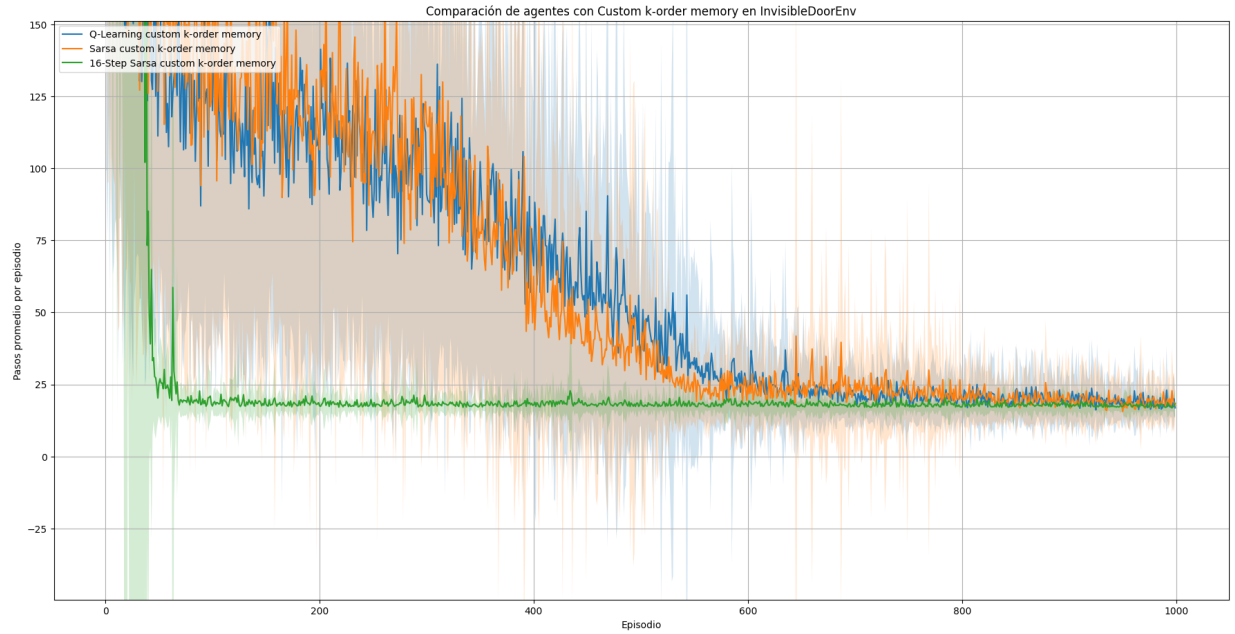

Figure 8: Promedio de 30 ejecuciones con agentes con *custom k-order memory* en **InvisibleDoorEnv**, cada una con 1000 episodios. Vista con Zoom.

La memoria que mejor desempeño mostró fue la *custom k-order memory*, ya que permite al agente guardar explícitamente información clave del pasado (en este caso haber apretado el botón), lo que le da una ventaja

crítica en entornos parcialmente observables como **InvisibleDoorEnv**. Esta memoria logra una representación más relevante del estado sin depender del límite temporal fijo como en *k-order* ni del aprendizaje complejo de escritura como en *binary*. En los gráficos, se ve cómo todos los algoritmos con esta memoria convergen rápidamente a políticas óptimas (menos de 25 pasos por episodio), mientras que con *k-order memory* solamente **16-step SARSA** tiene un buen desempeño, y con *binary memory* el aprendizaje es más errático por la dificultad de manipular el bit correctamente. Finalmente, los agentes sin memoria fallan completamente en aprender una política efectiva, pues no pueden saber el estado real de la puerta.

El algoritmo más efectivo fue **16-step SARSA**, que consistentemente superó a **Q-Learning** y **SARSA** en entornos con memoria estructurada (*k-order* y *custom*). Este algoritmo aprovecha la información de múltiples pasos, lo que le permite suavizar la señal de aprendizaje y capturar mejor las consecuencias de acciones como apretar el botón, especialmente útil en ambientes con recompensas tardías. En el caso de *k-order memory*, esto es muy importante, ya que aunque la memoria de orden 2 no siempre permite inferir directamente el estado real de la puerta, **16-step SARSA** puede aprender políticas útiles al propagar información hacia el pasado (hasta 16 pasos atrás), lo que le permite recompensar apretar el botón en el pasado, incluso si la observación inmediata de haberlo hecho es ambigua. Por eso logra converger rápidamente, mientras que **SARSA** y **Q-Learning**, que solo usan una señal inmediata, no logran resolver la tarea y su rendimiento sigue empeorando con el tiempo.