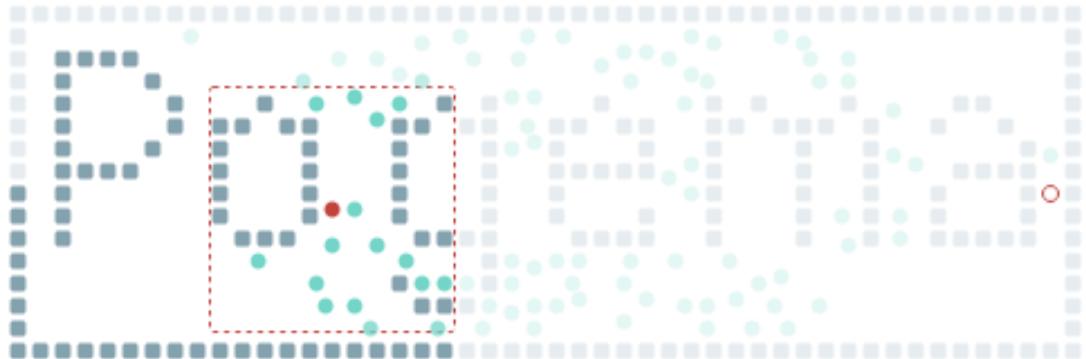


# Pràctica 3 (Laboratorio)



Lluc Santamaria Riba  
Francisco Badia Laguillo  
José Durán Foix

<b>Introducción.....</b>	<b>3</b>
<b>Experimento 1.....</b>	<b>4</b>
Solución de Pareto.....	6
Solución de Nash.....	14
Solución de Welfare.....	20
Solución de Minimax.....	25
Conclusiones.....	26
<b>Experimento 2.....</b>	<b>27</b>
Conclusión.....	28
<b>Experimento 3.....</b>	<b>29</b>
Conclusión.....	34
<b>Experimento 4.....</b>	<b>35</b>
Tres agentes.....	35
Mapa de 8x8.....	35
Conclusión.....	36
<b>Experimento 5.....</b>	<b>37</b>
Conclusión.....	37
<b>Extra:.....</b>	<b>38</b>
Introducción y objetivos:.....	38
Hipótesis:.....	38
Diseño de los experimentos:.....	39
Parámetros fijos:.....	39
Parámetros que buscamos optimizar:.....	40
Uso de wandb:.....	40
Resultados:.....	40
Conclusiones y recomendaciones para futuros experimentos:.....	41
<b>Conclusiones.....</b>	<b>42</b>

# Introducción

Para afrontar esta práctica, hemos dividido el trabajo en diversos experimentos. Cada experimento aborda una o más de las cuestiones planteadas en el enunciado de la práctica. La ejecución de cada experimento es replicable y está documentada en el fichero README. Cada experimento parte de unas hipótesis y suposiciones planteadas en base al funcionamiento teórico de los algoritmos y las características del entorno. Una vez obtenidos los resultados, se discuten las hipótesis y se dan posibles recomendaciones de cómo entrenar agentes para entornos similares.

Se ha utilizado la librería de Wandb para facilitar la experimentación de los algoritmos con sus hiperparámetros. Se ha tenido que separar el funcionamiento en dos entornos virtuales distintos ya que Pogema 1.2.2 tiene dependencias con Pydantic < 2.0 mientras que Wandb tiene dependencias con Pydantic  $\geq 2.0$ . La ejecución de Wandb inicializa la ejecución de Pogema, los cuales se pasan ficheros serializados para comunicarse.

Además, en la mayoría de experimentos hemos implementado un *learning rate decay* y un *epsilon decay* siguiendo una función exponencial decreciente hasta alcanzar los valores mínimos. Por un lado, el *learning rate decay* permite que un agente tienda a ser más conservador en cuanto a la actualización de su política. Las actualizaciones de política se estabilizan a medida que un agente se vuelve más experto sobre su entorno. Esto es especialmente útil en el entorno parcialmente observable, donde hay que evitar sobreajustar los valores Q con experiencias recientes. Por otro lado, el *epsilon decay* permite un buen ajuste entre exploración y explotación. Por defecto, las recompensas de los agentes son *sparse* y retardadas (+1 si alcanza el objetivo, -0,01 en caso contrario), por lo que es importante asegurar una buena exploración al principio que se vaya estabilizando a explotación en cuanto el agente ya tenga conocimiento de una política considerable.

El valor principal utilizado para comparar los rendimientos es el promedio de las recompensas obtenidas en las evaluaciones de todos los *epochs*. Esto permite beneficiar a las configuraciones de hiperparámetros que permiten a los agentes aprender rápidamente a la vez que beneficia las configuraciones que obtienen mejores resultados. En esta práctica, una ejecución de un experimento puede demorar más de 3 horas.

# Experimento 1

Este primer experimento responde a las preguntas del enunciado:

1. ¿Qué parámetros, y con qué valores, hacen que el algoritmo converja mejor en cuanto a recompensa, individual y colectiva? ¿Y en cuanto a tiempo de entrenamiento?
2. ¿Cómo se comportan los agentes cuando están entrenados para optimizar su comportamiento respecto de cada concepto de solución incluido (minimax, Nash, welfare, Pareto)? ¿Para qué tipo de coordinación es más adecuado cada uno (intereses comunes, conflictivos, mixtos)?

Estudiaremos las configuraciones de hiperparámetros que hacen que el algoritmo tenga un mejor comportamiento (obteniendo mejores resultados) para cada uno de los conceptos de solución ya implementados. Es decir, para el concepto de solución óptimo de Pareto, encontraremos la mejor configuración y estudiaremos cómo se comportan los agentes en base a los fundamentos teóricos del concepto de solución. Y lo mismo para Minimax, Nash y Welfare. En cada concepto de solución estudiado veremos cómo se comportan los agentes una vez entrenados para optimizar su comportamiento.

Para los cuatro conceptos de solución usaremos los mismos rangos de parámetros e hiperparámetros. Algunos de los parámetros ya han sido fijados para simplificar el estudio del experimento (muchos de ellos son los que venían por defecto, se ha justificado por qué nos sirven). Las configuraciones que usaremos son:

1. Fijamos el tamaño del mapa a 4x4. A priori, no debería tener ninguna influencia significativa en los hiperparámetros de los agentes, y aumentarlo solo provocaría que el estudio fuese más costoso en recursos. Además, estudiar la escalabilidad del sistema corresponde a un experimento posterior.
2. La representación de los estados es la dada por defecto, con un total de 1024 estados (4 bits por los obstáculos, 4 bits por los agentes alrededor y 2 bits por el objetivo).
3. Fijamos el número de agentes a 2. Utilizar dos agentes permite estudiar el sistema desde un enfoque multiagente. Además, el mapa es muy pequeño para un número más elevado de agentes, pudiendo dificultar el estudio dando lugar a no estacionariedad o imposibilidad de alcanzar un objetivo.
4. Fijamos el número de mapas para entrenar a 10. En un entorno de 4x4, no hay muchas combinaciones de mapas distintos, por lo que creemos que diez mapas es un número representativo en el que los agentes se enfrentan a situaciones suficientemente variadas.
5. Fijamos el número de epochs a 200. Con el código proporcionado inicialmente que usaba 200 epochs, hemos visto que era más que suficiente para observar tendencias. Si un agente tiene una configuración de parámetros óptima, entonces 200 epochs debería ser más que suficiente para que el agente alcance una política óptima (lo veremos más adelante).

6. Fijamos la densidad de obstáculos a 0.1. Tener muchos obstáculos no resulta provechoso para este estudio, especialmente en un entorno tan pequeño. Sin embargo, tener algún obstáculo permite observar cómo se comportan los agentes ante él. Por eso, 0.1 es un valor pequeño pero suficiente
7. Fijamos el radio de observación a 1. En un entorno tan pequeño tener un radio más grande sería como casi tener observabilidad total. Con un radio de 1 los agentes ven casi la mitad del mapa, lo cual permite introducirse en los conceptos de observabilidad parcial.
8. Fijamos el número de episodios por *epoch* a 10. En cada *epoch* hacemos un promedio de los resultados obtenidos. Un número muy elevado de episodios puede resultar costoso, y con 10 episodios ya es suficiente para obtener una muestra representativa.
9. Fijamos el número máximo de pasos por episodio a 16. En cada episodio se realiza una ejecución de los agentes en el entorno. El número máximo de pasos limita el tiempo disponible para el agente para alcanzar su objetivo. Creemos que 16 es un número suficiente para el entorno de 4x4, ya que permite al agente pasearse por todo el entorno en caso que fuera necesario.
10. Los valores de *gamma* que probaremos son 0.5, 0.95, 0.97, 0.99, 0.999. Las recompensas del entorno son *sparse* (solo hay un reward positivo cuando el agente alcanza el objetivo). Esto hace que aunque el agente tenga una visión más cortoplacista, siempre tienda a ir hacia el objetivo, ya que es el único reward positivo. No obstante, como hipótesis creemos que mayores valores de *gamma* harán que el agente sea más directo para alcanzar su objetivo, ya que tiene una visión más hacia el futuro.
11. Para el valor inicial de *epsilon* probaremos los valores 0.4, 0.8, 0.9, 1.0. Para los valores mínimos de *epsilon* probaremos los valores 0.01, 0.05, 0.1, 0.2, 0.3. Para el *epsilon decay* probaremos los valores 0.7, 0.95, 0.99, 0.999. Creemos que la exploración es muy importante sobre todo al inicio, ya que las recompensas son *sparse* y retardadas. Como hipótesis creemos que valores mayores de *epsilon* inicial son mejores (garantizan exploración al inicio). Para el valor mínimo de *epsilon* creemos que un valor intermedio es el mejor, porque aunque el agente ya tenga una política buena, hay que garantizar un mínimo de exploración por si el agente no había explorado aún otro camino de estados. Para el *epsilon decay* creemos que el valor tiene que garantizar que *epsilon* tienda al mínimo a lo largo de los 200 *epochs* (sin alcanzar el mínimo muy rápido o nunca alcanzarlo).
12. Para el valor inicial de *alpha* probaremos los valores 0.01, 0.05, 0.1, 0.3. Para los valores mínimos de *alpha* probaremos los valores 0.0, 0.0001, 0.001, 0.01. Para el *alpha decay* probaremos los valores 0.7, 0.95, 0.99, 0.999. Creemos que garantizar aprendizaje al principio es importante, y también debe serlo al final (de hecho el algoritmo por defecto no utilizaba *alpha decay* y los agentes encontraron una política correcta). Por esto, como hipótesis creemos que valores mayores (pero no excesivos) de *alpha* inicial y *alpha* mínimo mejor irá para que el agente pueda aprender garantizando una estabilidad. De forma análoga al *epsilon decay*, creemos que el *alpha decay* debe garantizar una buena convergencia del valor inicial al valor mínimo a lo largo del estudio del experimento.

## Solución de Pareto

En este estudio fijamos en todos los agentes el concepto de solución de óptimo de Pareto. Desde un punto de vista teórico, creemos que el entorno conlleva a los agentes a tener intereses mixtos. Esto es porque cada agente tiene su propio objetivo a alcanzar, por lo que un aumento en la recompensa de un agente no va asociado a un aumento en la recompensa de otro agente (aunque tampoco debería ir asociado a una pérdida). No obstante, los dos agentes siempre preferirían no colisionar para evitar acciones no útiles. Como hipótesis creemos que usar óptimos de Pareto permitirá a los agentes encontrar buenas políticas como soluciones al problema, aunque no las mejores, ya que creemos que los agentes mostrarán un comportamiento egoísta.

Usando Wandb, el gráfico de configuraciones de hiperparámetros obtenido es el de la Figura 1.

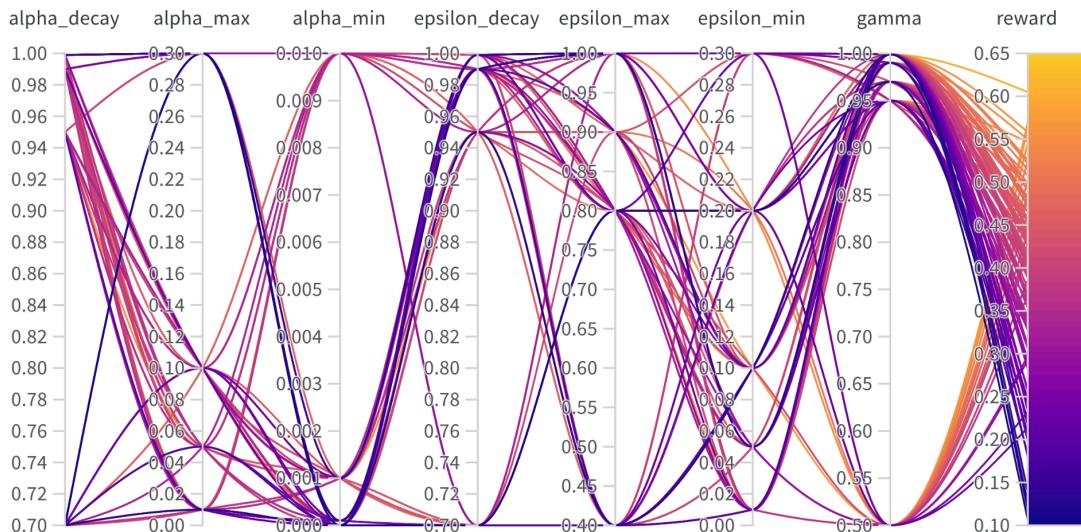


Figura 1: Configuraciones de hiperparámetros en el estudio de solución de Pareto.

Hemos usado como recompensa resultante un promedio de las recompensas obtenidas individualmente (es decir, la suma total dividida entre el número de agentes).

En la mejor de las configuraciones, los agentes reciben una recompensa media de 0.6. La configuración que ha llevado a esta recompensa es *alpha decay* de 0.95, *alpha* inicial de 0.01, *alpha* mínimo de 0.01, *epsilon decay* de 0.7, *epsilon* inicial de 1, *epsilon* mínimo de 0.3 y *gamma* de 0.999. Notar que el *alpha* inicial es igual al mínimo, por lo que no hay *decay*.

Las correlaciones de los hiperparámetros con el rendimiento promedio de los agentes puede observarse en la Figura 2.

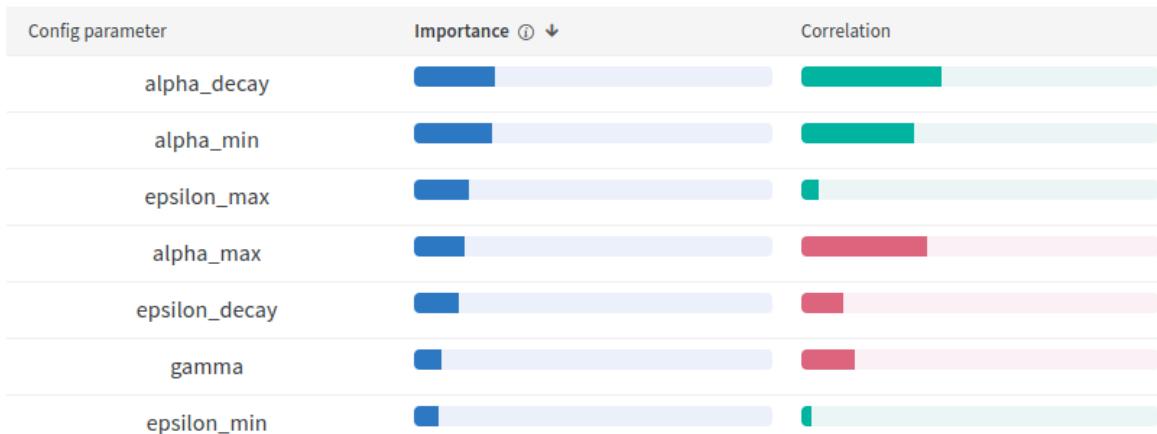


Figura 2: Correlaciones de los hiperparámetros en el estudio de solución de Pareto.

Vemos como tener un *alpha* mínimo mayor presenta mejores resultados, sumado a un *alpha* inicial (*alpha\_max*) pequeño y un *alpha decay* grande. Esto parece indicar que el algoritmo funciona mejor con valores de *alpha* pequeños pero estables, lo que contradice nuestra hipótesis de aprender mucho al inicio y buscar estabilidad al final. Parece que los agentes quieren mantener un aprendizaje estable a lo largo de toda la experimentación, aunque sea en una medida pequeña.

Por otro lado, vemos que la mejor combinación para *epsilon* es usar valores elevados para el valor inicial sumado de valores también elevados para el valor mínimo y con un decrecimiento lo menor posible. Aunque las correlaciones son moderadas. Esto confirma parte de nuestra hipótesis: los agentes quieren mantener una exploración elevada al inicio, aunque también la quieren al final por lo que prefieren valores mínimos mayores. Lo que nos ha sorprendido es que los agentes prefieren una convergencia rápida a los valores mínimos, indicando que demasiada exploración al inicio puede resultar contraproducente.

Finalmente vemos cómo la *gamma* presenta una correlación negativa. No obstante, hemos visto cómo el valor más elevado de recompensa se obtiene con la mayor *gamma*. También nos fijamos que la importancia del parámetro es reducida. Esto parece indicar que el valor de *gamma* no es muy importante, lo cual confirma parte de nuestra hipótesis donde explicamos que el factor de tener recompensas *sparse* y retardadas hace que no tenga tanta importancia el valor de *gamma*. Aún así, niega nuestra parte de que mayores valores dan mejores resultados.

Para estudiar los resultados de la experimentación, nos fijamos en las 4 configuraciones de hiperparámetros que mejores rendimientos han obtenido, como puede verse en la Figura 3.

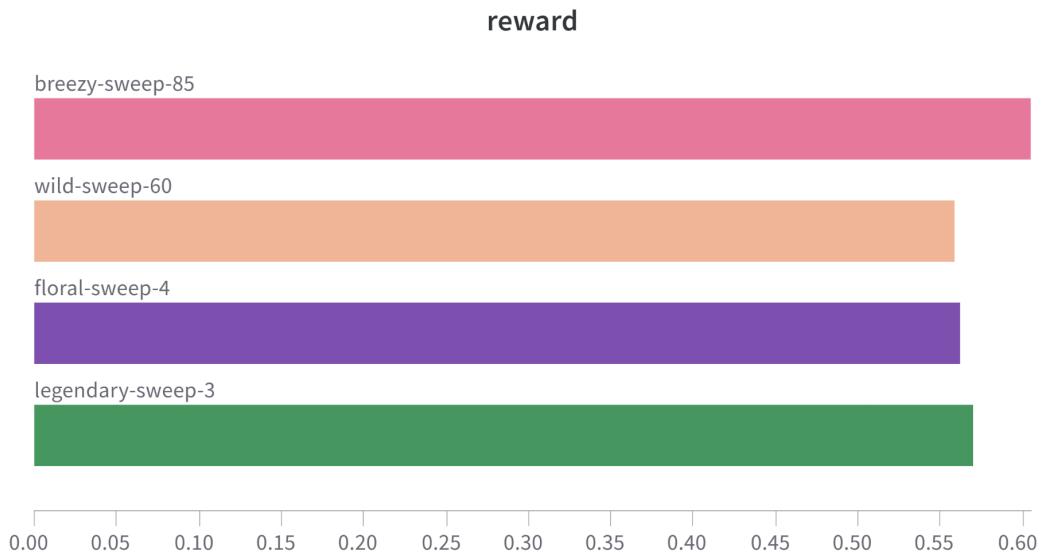


Figura 3: Configuraciones con mejor rendimiento en el estudio de solución de Pareto.

En la Figura 4, podemos observar las recompensas obtenidas para estas configuraciones durante la evaluación. Recordamos que la métrica de recompensa que se utiliza como objetivo a maximizar es la media entre todos los *epochs*. Es por esto que se observan valores ligeramente superiores al 0.6.

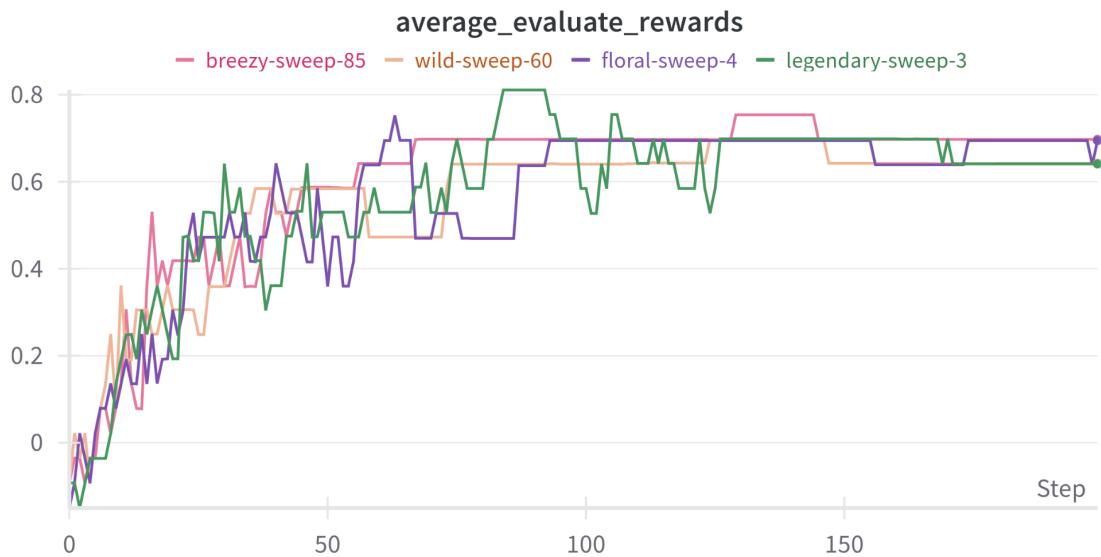


Figura 4: Recompensas obtenidas para las mejores configuraciones en el estudio de solución de Pareto.

Con la función de recompensa del entorno (+1 al alcanzar el objetivo y -0,01 en caso contrario), la recompensa obtenida está fuertemente vinculada al ratio de éxito (que proporción de agentes alcanzan su objetivo). En la Figura 5 se puede observar el ratio de éxito para la evaluación de los agentes. Este ratio parece indicar que con una política óptima, 6 de cada 10 veces se alcanza el objetivo.

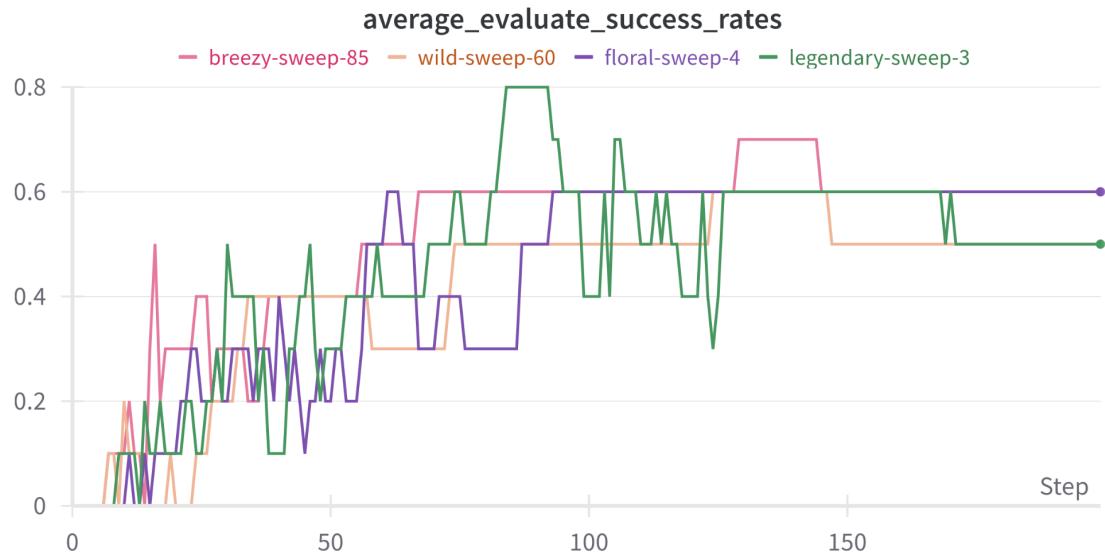


Figura 5: Ratio de éxito para las mejores configuraciones en el estudio de solución de Pareto.

El ratio de éxito también puede utilizarse como indicador de la calidad del entrenamiento. En este caso donde los agentes utilizan conceptos de solución de Pareto, hay un 60% de éxito. Dicho de otra forma, el entrenamiento ha alcanzado el 60% del potencial de los agentes.

Podemos también estudiar el error de diferencias de los algoritmos utilizados por los agentes. En la Figura 6 se puede observar el promedio de suma de errores de los agentes (suma de errores entre número de agentes) mientras que en la Figura 7 y la Figura 8 se pueden observar la suma de errores de los agentes por separado.

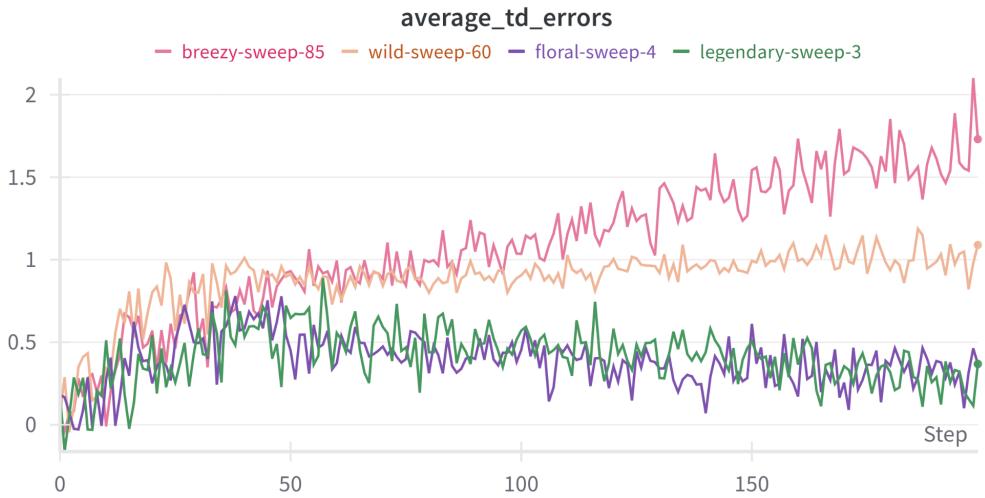


Figura 6: Diferencias temporales promedio para el estudio de Pareto.

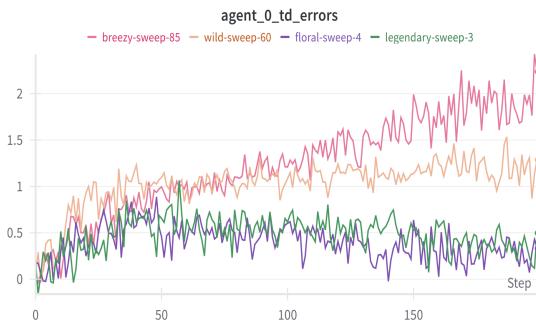


Figura 7: Diferencias temporales para el primer agente.



Figura 8: Diferencias temporales para el segundo agente.

Podemos ver como las diferencias temporales promedio parecen seguir diferentes tendencias para diferentes configuraciones de hiperparámetros. Por ejemplo, la configuración de color rosa, la cual obtuvo mejor recompensa, muestra que las diferencias temporales tienden a ser mayores. Esto podría conllevar a que la política del agente no se estabilice, aunque en la Figura 5 del ratio de éxito parece indicar que está estable, por lo que las diferencias temporales no parecen influir en el éxito de los agentes. También hemos añadido las diferencias temporales de los agentes por separado para mostrar como son casi idénticas. En el estudio de agentes con diferentes algoritmos o conceptos de solución esto podría no ser así.

Finalmente, en la Figura 9 podemos ver cómo los agentes empiezan necesitando unos 14 pasos por ejecución y terminan necesitando solamente 6 pasos por ejecución, por lo que el parámetro fijado en 16 pasos máximo ha sido justo y suficiente.

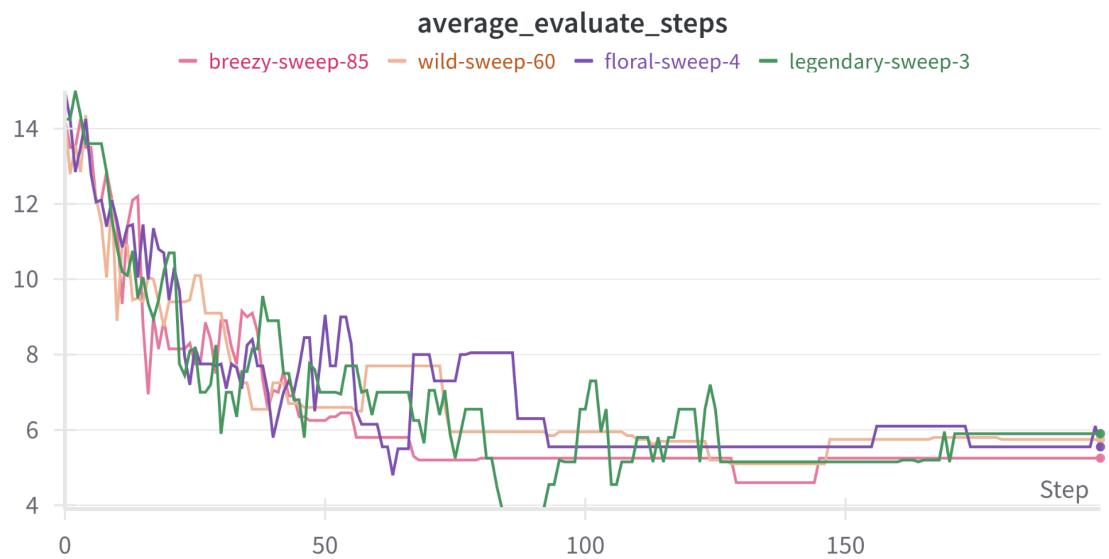


Figura 9: Número promedio de pasos para el estudio de Pareto.

También podemos decir que los 200 *epochs* fijados es suficiente, porque todas las gráficas muestran estabilizaciones antes de alcanzar los 200 *epochs*.

El tiempo total de este experimento con soluciones de Pareto ha sido de 2 horas. La media del tiempo de ejecución ha sido de 95.5 segundos por configuración y la desviación estándar ha sido de 9.6 segundos. En la Figura 10 podemos observar la distribución del tiempo de ejecución para todas las configuraciones.

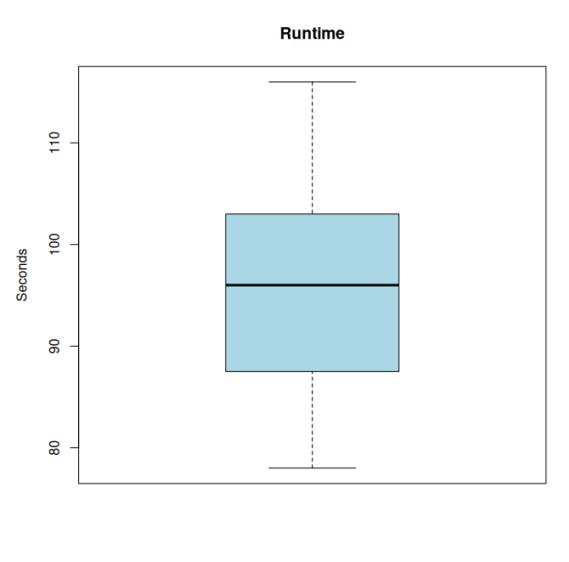


Figura 10: Distribución del tiempo de ejecución para el estudio de Pareto.

Para responder a la pregunta de cómo se comportan los agentes cuando están entrenados para optimizar su comportamiento utilizando el concepto de solución de Pareto, podemos ayudarnos de distintos gráficos de la evaluación de estas 4 mejores configuraciones de hiperparámetros. Notar que durante el proceso de evaluación los agentes aún se están entrenando, pero como hemos visto anteriormente, a partir de los 100 *epochs* aproximadamente los agentes ya muestran un fuerte aprendizaje que les permite obtener buenos resultados (ver Figura 4 y Figura 5 donde se muestran las recompensas obtenidas y el ratio de éxito respectivamente).

Por un lado, en la Figura 11 y 12 puede observarse la recompensa media obtenida por cada agente en la fase de evaluación. Comparando las dos, puede verse como hay ciertas zonas donde un agente obtiene buenas recompensas mientras otro no obtiene recompensas tan buenas. Por ejemplo, a partir del *epoch* 170, el agente 0 no tiene recompensas excelentes en la configuración *floral-sweep-4* y tiene buenas recompensas en la configuración *legendary-sweep-3* mientras que el agente 1 tiene excelentes recompensas para *floral-sweep-4* pero no tan buenas para *legendary-sweep-3*. Aunque también cabe destacar que no ha sido un *tradeoff*: el agente 0 ya venía con la misma tendencia en las recompensas desde el *epoch* 125.

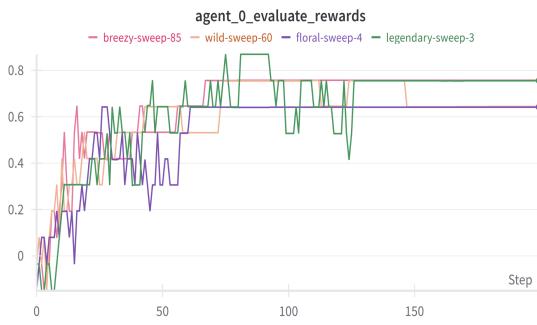


Figura 11: Recompensas en la evaluación para el primer agente.

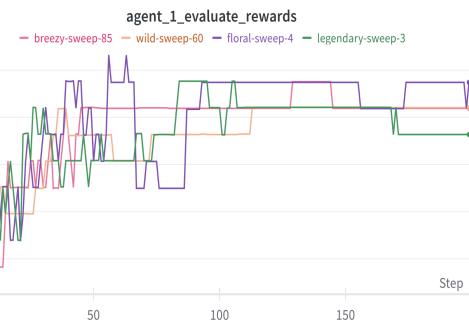


Figura 12: Recompensas en la evaluación para el segundo agente.

Desde un punto de vista teórico, como se ha explicado al inicio de este apartado con concepto de solución de Pareto, un aumento en la recompensa de un agente no debería ir asociado a una pérdida para el otro. Esto es porque la implementación de Pareto busca una estrategia mixta en el conjunto de soluciones de Pareto, con lo cual esta estrategia debería ser óptima para ambos. Creemos que estas diferencias son más bien producto de la aleatoriedad.

En las Figuras 13 y 14, puede observarse el promedio de pasos por ejecución necesitados por agente en la evaluación. En este ya no se observa esta relación entre agentes. Es decir, el número de pasos que necesita un agente para alcanzar su objetivo no depende del número de pasos que necesita el otro.



Figura 13: Pasos por ejecución en la evaluación para el primer agente.



Figura 14: Pasos por ejecución en la evaluación para el segundo agente.

Para concluir este apartado, podemos decir que el concepto de solución de óptimo de Pareto obtiene buenos resultados (recompensa máxima de 0.6 y ratio de éxito del 60%) y unos resultados igualitarios para los diferentes agentes, lo cual es consistente con la base teórica explicada previamente.

## Solución de Nash

En este estudio fijamos en todos los agentes el concepto de solución de equilibrio de Nash.

Siguiendo el mismo razonamiento que en apartado de solución de Pareto, donde se ha explicado que creemos que los agentes tienen intereses mixtos, creemos que con soluciones de Nash se obtendrán resultados ligeramente inferiores (desde el punto de vista global usando recompensa media obtenida). Como se vio con el Dilema del Prisionero, un equilibrio de Nash puede ser ineficiente desde un hipotético control centralizado, aunque para los agentes sea una estrategia que no querrían cambiar aunque supieran la estrategia del contrincante. No obstante, puede resultar más fácil encontrar equilibrios de Nash en comparación a los óptimos de Pareto, por lo que creemos que habrá menos configuraciones que obtengan malos resultados. La Figura 15 muestra la recompensa promedio obtenida para cada una de las configuraciones de hiperparámetros.

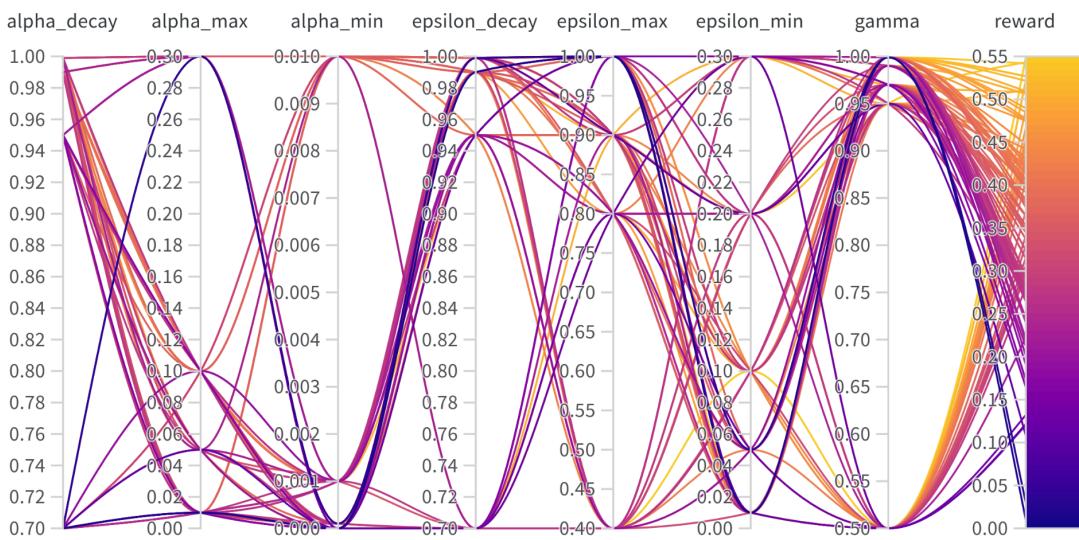


Figura 15: Configuraciones de hiperparámetros en el estudio de solución de Nash.

Efectivamente, no se ha alcanzado la recompensa de 0.6 que se obtuvo con óptimo de Pareto. Además, podemos ver como no hay tantas configuraciones que obtengan resultados de los rangos inferiores en comparación con el concepto de solución de Pareto.

En la mejor de las configuraciones, los agentes reciben una recompensa media de 0.55. La configuración que ha llevado a esta recompensa es *alpha decay* de 0.99, *alpha* inicial de 0.3, *alpha* mínimo de 0.01, *epsilon decay* de 0.99, *epsilon* inicial de 0.4, *epsilon* mínimo de 0.1 y *gamma* de 0.99.

Las correlaciones de los hiperparámetros con el rendimiento promedio de los agentes puede observarse en la Figura 16.

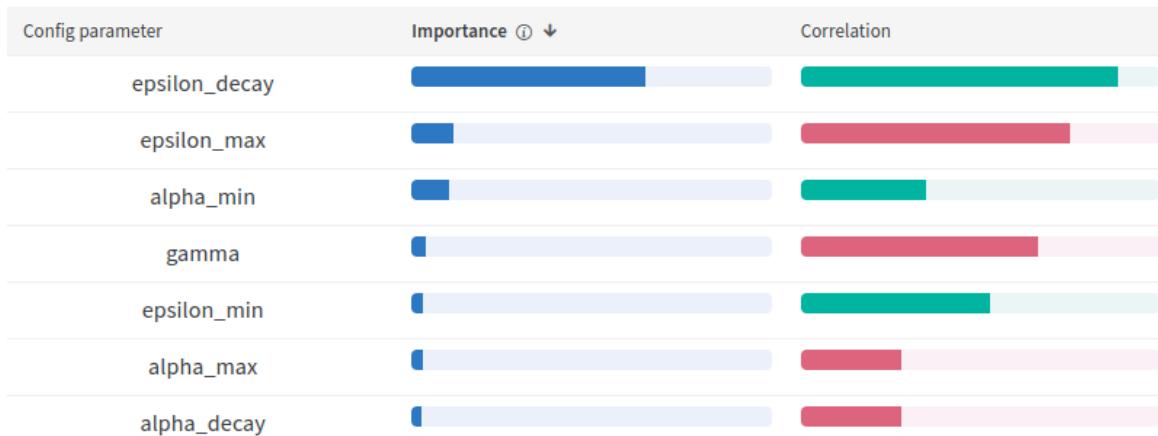


Figura 16: Correlaciones de los hiperparámetros en el estudio de solución de Nash.

Con este concepto de solución, vemos cómo de nuevo la *gamma* presenta una correlación fuertemente negativa aunque la importancia de la correlación es moderada. De nuevo, contrasta con la mejor de las configuraciones la cuál ha usado un *gamma* de 0.99, con lo que no tenemos suficiente evidencia para refutar nuestra hipótesis sobre los valores de *gamma*.

Para el *epsilon*, hay una correlación positiva y de mucha importancia en el *epsilon decay*, por lo que la exploración suele converger rápidamente a explotación. Además, la *epsilon inicial* presenta una correlación fuertemente negativa, lo que significa que ya de partida los agentes prefieren explotar antes que explorar. Finalmente, la *epsilon* mínima presenta una correlación positiva, aunque de una importancia muy moderada. Esto refuta nuestra hipótesis donde defendimos que los agentes buscarían una fuerte exploración que progresivamente converge a explotación. Parece que los agentes prefieren explorar poco pero suficiente a lo largo de todos los *epochs*.

Por otro lado, vemos como el *alpha* inicial y el *alpha decay* presentan correlaciones negativas aunque de importancia casi nula mientras que el *alpha* mínimo presenta una correlación positiva. Esto sugiere que los agentes prefieren una *alpha* que siempre garantice un aprendizaje suficiente a lo largo de la experimentación. De forma similar que con el *epsilon*, se rechaza nuestra hipótesis inicial para *alpha*.

De nuevo, para estudiar los resultados de la experimentación, nos fijamos en las 4 configuraciones de hiperparámetros que mejores rendimientos han obtenido, como puede verse en la Figura 17.



Figura 17: Configuraciones con mejor rendimiento en el estudio de solución de Nash.

Para evaluar la calidad del entrenamiento, podemos ver el ratio de éxito promedio y compararlo con las recompensas obtenidas. La Figura 18 corresponde al ratio de éxito en evaluación mientras que la Figura 19 corresponde a las recompensas promedias obtenidas en la evaluación.



Figura 18: Ratio de éxito en la evaluación.



Figura 19: Recompensas obtenidas en la evaluación.

Mientras que con el concepto de solución de Pareto se obtiene un ratio de éxito más o menos estable alrededor del 0.6, en el concepto de solución de Nash los valores varían entre el 0.8 y 0.4. Además, algunas de las recompensas para estas configuraciones superan la recompensa media de 0.55. Este resultado indica que la política óptima se obtiene de forma más progresiva en comparación al concepto de solución de Pareto, pero termina obteniendo iguales o mejores recompensas. Sin embargo, la media del ratio de éxito podemos considerar que continúa siendo de 0.6.

Para obtener una mejor imagen de lo que sucede en la política de los agentes, podemos analizar la media de errores por diferencias temporales en la Figura 20.

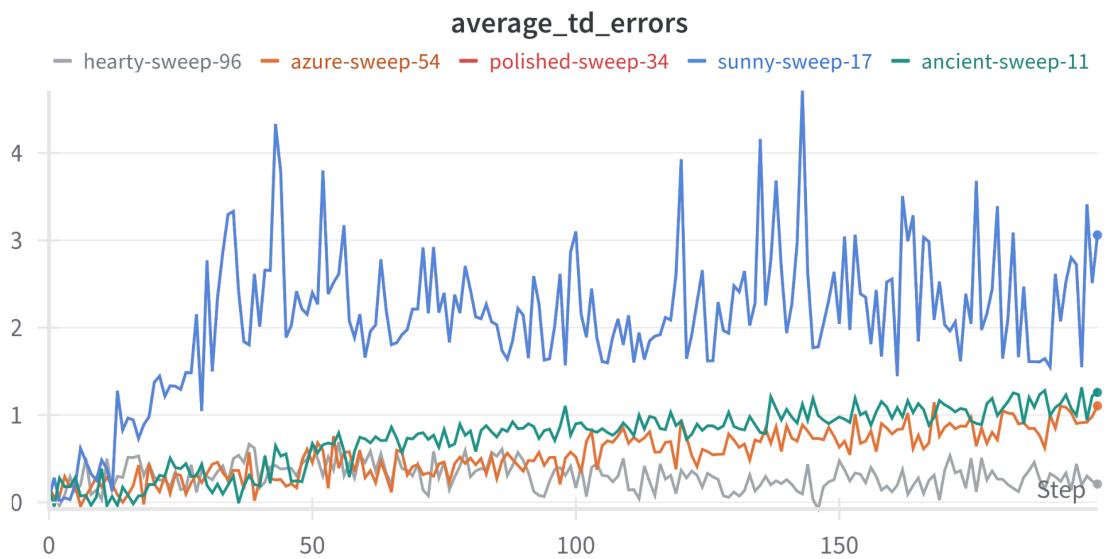


Figura 20: Diferencias temporales promedio para el estudio de Nash.

El resultado es similar respecto al estudio del concepto de solución de Pareto, donde los valores pertenecen al rango de 0 a 1 excepto la configuración *sunny-sweep-17*. De nuevo, no muestran una tendencia a 0, por lo que las mejores configuraciones prefieren aprender a lo largo de todos los *epochs*.

Respecto al número de *epochs* y pasos por ejecución, los valores de 200 y 16 han sido nuevamente suficientes.

El tiempo total de este experimento con soluciones de Nash ha sido de 1 hora. La media del tiempo de ejecución ha sido de 62.1 segundos por configuración y la desviación estándar ha sido de 4.75 segundos. En la Figura 21 podemos observar la distribución del tiempo de ejecución para todas las configuraciones.

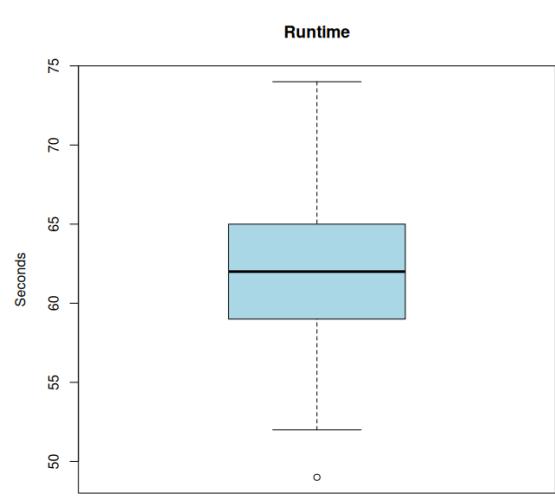


Figura 21: Distribución del tiempo de ejecución para el estudio de Pareto.

Para responder a la pregunta de cómo se comportan los agentes cuando están entrenados para optimizar su comportamiento utilizando el concepto de solución de Nash, seguiremos los mismos pasos que con el concepto de solución de Pareto.

Las figuras 22 y 23 comparan las recompensas en la evaluación entre los dos agentes. Las figuras 24 y 25 comparan los pasos por ejecución de la evaluación entre los dos agentes.

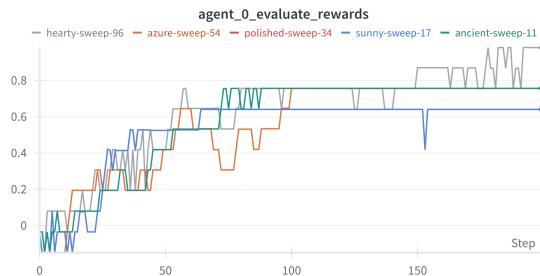


Figura 22: Recompensas en la evaluación para el primer agente.

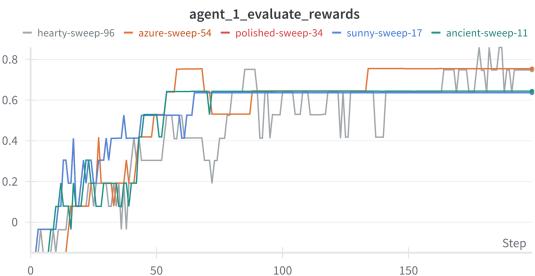


Figura 23: Recompensas en la evaluación para el segundo agente.



Figura 24: Pasos por ejecución en la evaluación para el primer agente.

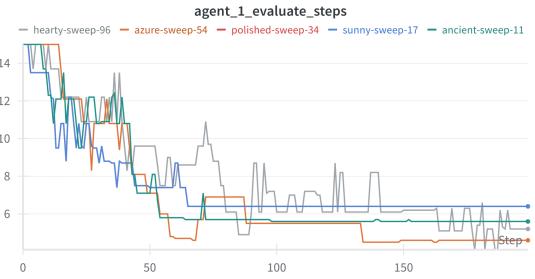


Figura 25: Pasos por ejecución en la evaluación para el segundo agente.

De nuevo, los gráficos parecen indicar que los valores de un agente son independientes de los valores del otro agente. El concepto de solución de equilibrios de Nash garantiza un equilibrio donde los dos agentes preferirían mantener su estrategia, por lo que los agentes simplemente se comportan siguiendo su estrategia sin perjudicarse mutuamente.

Para concluir este apartado, podemos decir que el concepto de solución de Nash obtiene buenos resultados (recompensa máxima de 0.5 y ratio de éxito del 60%). Unos resultados bastante parecidos al óptimo de Pareto. Queremos destacar que para alcanzar una política que conlleve a buenas recompensas, el concepto de solución de Pareto converge más rápido que el concepto de solución de Nash, lo que hace que las recompensas promedio sean inferiores.

## Solución de Welfare

En este estudio fijamos en todos los agentes el concepto de solución de Welfare. La solución de Welfare busca maximizar la suma de utilidades. Uno de los inconvenientes de Welfare es que no da ninguna garantía sobre la equidad, pudiendo un agente obtener recompensas elevadas a costa de perjudicar a otro agente. Sin embargo, como ya hemos explicado, los agentes no tienen intereses conflictivos ni es un juego de suma cero, por lo que creemos que maximizar la suma de utilidades conlleva a una maximización individual. Por esto, como hipótesis creemos que los resultados serán muy similares al estudio utilizando óptimos de Pareto. En la Figura 26 se pueden observar las recompensas obtenidas para diferentes configuraciones.

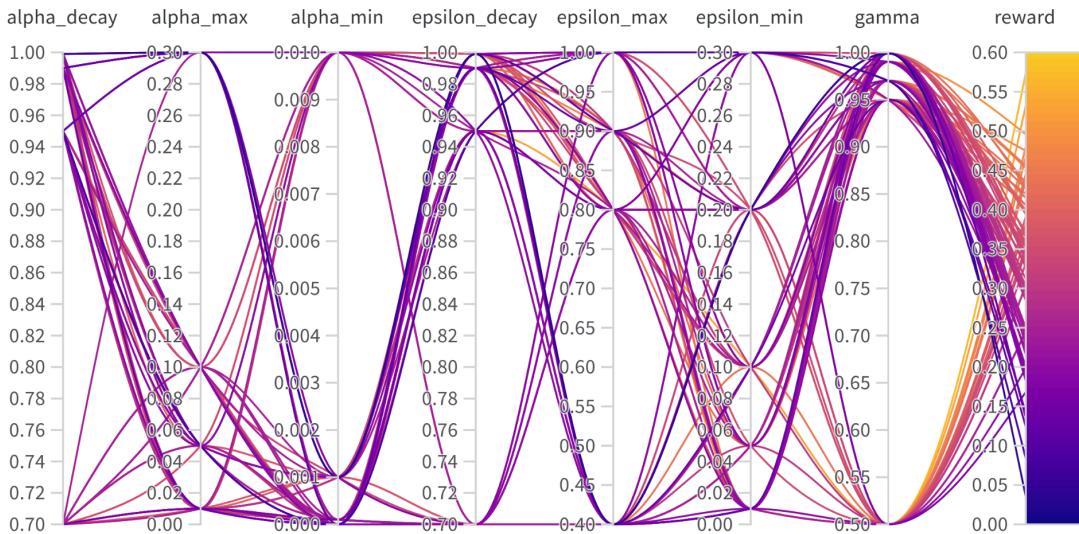


Figura 26: Configuraciones de hiperparámetros en el estudio de solución de Welfare.

Los rangos de recompensas obtenidas son muy similares a los del estudio de solución de Pareto. En la mejor de las configuraciones, los agentes reciben una recompensa media de 0.57. La configuración que ha llevado a esta recompensa es *alpha decay* de 0.999, *alpha* inicial de 0.3, *alpha* mínimo de 0.0001, *epsilon decay* de 0.99, *epsilon* inicial de 0.8, *epsilon* mínimo de 0.2 y *gamma* de 0.5.

Las correlaciones de los hiperparámetros con el rendimiento promedio de los agentes puede observarse en la Figura 27.



Figura 27: Correlaciones de los hiperparámetros en el estudio de solución de Welfare.

En este concepto de solución no hay correlaciones demasiado elevadas. En cuanto al *gamma*, de nuevo la relación es negativa. Además esta vez la mejor configuración presenta un *gamma* de solamente 0.5. Esto rechaza por completo nuestra hipótesis al respecto de *gamma*. Parece que los agentes tienen más aversión por las recompensas inmediatas.

Respecto al *epsilon*, está vez sí que los agentes prefieren valores iniciales elevados que convergen progresivamente a valores mínimos que continúen garantizando cierta exploración. Lo mismo para el *alpha* y el aprendizaje de los agentes. Esto confirma nuestras hipótesis formuladas inicialmente para *alpha* y *epsilon*.

De nuevo, para estudiar los resultados de la experimentación, nos fijamos en las 4 configuraciones de hiperparámetros que mejores rendimientos han obtenido, como puede verse en la Figura 28.



Figura 28: Configuraciones con mejor rendimiento en el estudio de solución de Welfare.

Para evaluar la calidad del entrenamiento, podemos ver el ratio de éxito promedio y compararlo con las recompensas obtenidas. La Figura 29 corresponde al ratio de éxito en evaluación mientras que la Figura 30 corresponde a las recompensas promedias obtenidas en la evaluación.

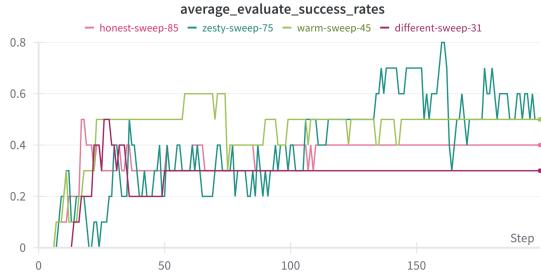


Figura 29: Ratio de éxito en la evaluación.

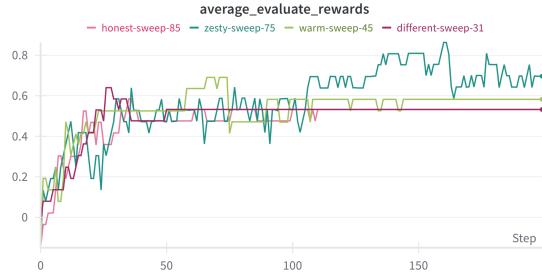


Figura 30: Recompensas obtenidas en la evaluación.

Los resultados para las recompensas siguen la misma tendencia que en el estudio de óptimo de Pareto: un rápido incremento y unas recompensas alrededor de 0.6. Para el ratio de éxito, en este caso presenta más varianza y una media de 0.5, ligeramente inferior a los resultados del estudio de óptimo de Pareto.

En la Figura 31 podemos ver como los errores por diferencias temporales son inferiores a los estudios previos, mostrando unos rangos que van de 0 a 1. Además, 3 de las 4 configuraciones muestran una tendencia decreciente. Esto se ajusta a nuestra hipótesis de aprender al inicio y ser más conservador a lo largo de la ejecución.

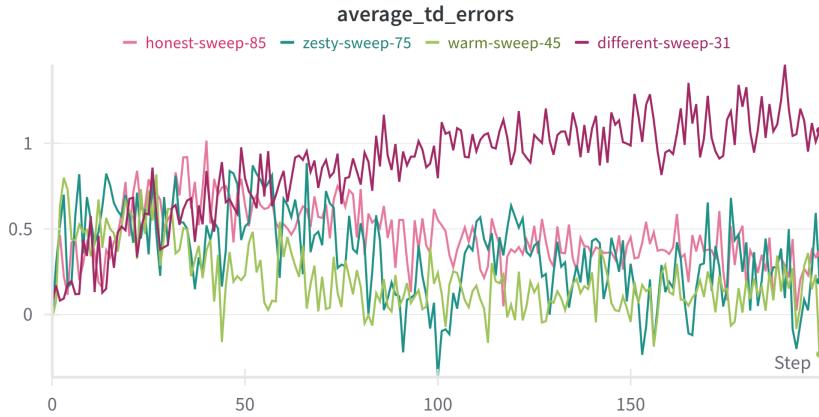


Figura 31: Diferencias temporales promedio para el estudio de Nash.

Respecto al número de *epochs* y pasos por ejecución, los valores de 200 y 16 han sido nuevamente suficientes.

El tiempo total de este experimento con soluciones de Welfare ha sido de 1 hora. La media del tiempo de ejecución ha sido de 66.3 segundos por configuración y la desviación estándar ha sido de 4.6

segundos. En la Figura 32 podemos observar la distribución del tiempo de ejecución para todas las configuraciones.

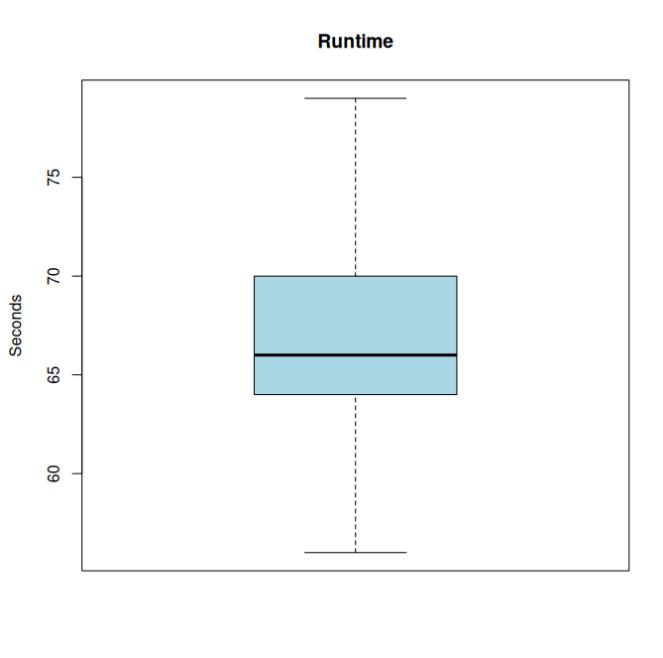


Figura 32: Distribución del tiempo de ejecución para el estudio de Welfare.

Para responder a la pregunta de cómo se comportan los agentes cuando están entrenados para optimizar su comportamiento utilizando el concepto de solución de Welfare, seguiremos los mismos pasos que con el concepto de solución de Pareto.

Las figuras 33 y 34 comparan las recompensas en la evaluación entre los dos agentes. Las figuras 35 y 36 comparan los pasos por ejecución de la evaluación entre los dos agentes.

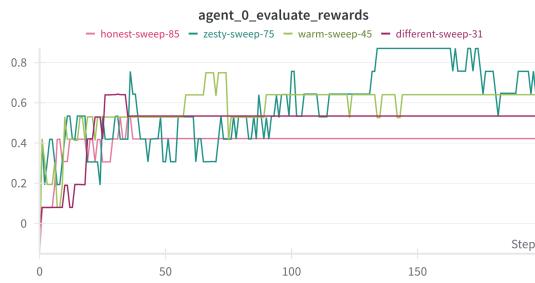


Figura 33: Recompensas en la evaluación para el primer agente.

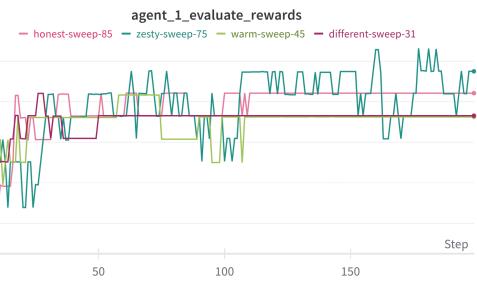


Figura 34: Recompensas en la evaluación para el segundo agente.



Figura 35: Pasos por ejecución en la evaluación para el primer agente.



Figura 36: Pasos por ejecución en la evaluación para el segundo agente.

De nuevo, los gráficos parecen indicar que los valores de un agente son independientes de los valores del otro agente. El concepto de solución de Welfare no parece hacer que los agentes desarrollen políticas competitivas. Los agentes prefieren mantener sus estrategias para maximizar la recompensa.

Para concluir este apartado, podemos decir que el concepto de solución de Welfare obtiene buenos resultados (recompensa máxima de 0.57 y ratio de éxito del 50%). Unos resultados casi equivalentes al estudio de solución de Pareto. Además, el tiempo de ejecución en este experimento es reducido, muy parecido al estudio de solución de Nash. Por lo que podemos decir que Welfare ha heredado la rapidez de Nash con la bondad de Pareto.

## Solución de Minimax

En este estudio fijamos en todos los agentes el concepto de solución de Minimax. Creemos que implementar Minimax como concepto de solución es altamente perjudicial para el rendimiento del sistema multiagente. Nos encontramos en un entorno donde los agentes tienen intereses mixtos o más bien egoístas. Un incremento en la recompensa de un agente no va asociado a un decremento en la de otro o viceversa, de hecho que el otro agente obtenga una recompensa es en parte beneficioso, ya que elimina más obstáculos en el camino. Es por esto que creemos que minimizar la máxima recompensa que puede obtener el otro agente será contraproducente. En la Figura 37 se puede observar las configuraciones para cada una de las configuraciones.

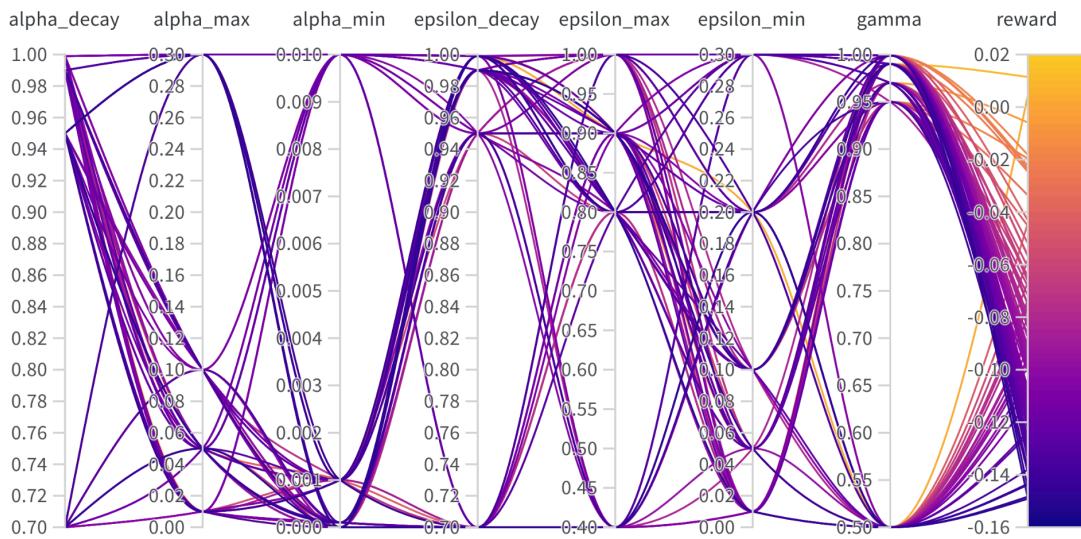


Figura 37: Configuraciones de hiperparámetros en el estudio de solución de Minimax.

En efecto, las recompensas obtenidas son nulas. De hecho, ningún agente consigue alcanzar el objetivo de forma sostenida, ya que las recompensas no superan el 0.02. Esto confirma nuestra hipótesis sobre recompensas pésimas usando Minimax. No tiene sentido continuar con este estudio porque los agentes no están aprendiendo ninguna política. Un agente no conseguirá encontrar su objetivo a partir de conseguir que el otro no lo consiga (y el otro hace lo mismo). Este algoritmo está pensado para juegos competitivos, con intereses conflictivos, como los juegos de suma cero.

## Conclusiones

Entrenar agentes en este entorno usando el concepto de solución de óptimo de Pareto, equilibrio de Nash o Welfare presenta unos resultados parecidos y buenos. Sin embargo, el concepto de solución de Minimax presenta unos resultados pésimos, ya que el entorno es para un tipo de intereses mixtos o más bien egoístas. Los tres primeros conceptos de solución hacen que los agentes tengan un buen comportamiento, sin perjudicarse mutuamente y siguiendo su propia estrategia. Como recomendación para entrenar agentes en entornos similares, creemos que antes de todo hay que verificar cuales son los intereses de los agentes (comunes, mixtos o conflictivos). Si el entorno es de intereses comunes, recomendamos utilizar óptimos de Pareto puesto que un incremento en la recompensa de un agente también beneficia a los demás. En caso de intereses mixtos recomendamos utilizar Welfare (hay que vigilar que se compla equidad) o Nash, puesto que presenta resultados buenos en tiempo de ejecución inferior o igual a los demás. Para los parámetros, hemos visto cómo puede variar entre conceptos de solución, por lo que recomendamos estudiar cada caso por separado (se puede partir de los resultados y correlaciones que hemos obtenido si el entorno es similar al nuestro).

## Experimento 2

Este segundo experimento responde a las preguntas:

1. ¿Qué ocurre si entrenamos a dos agentes con dos conceptos de solución diferentes? ¿Son capaces de coordinarse, o de conseguir recompensas individuales independientemente de la política del otro agente?

En el experimento anterior, hemos visto cómo el concepto de solución de Welfare es muy parecido al de Pareto, aunque el de Pareto mejora en recompensa y el de Welfare mejora en tiempo de ejecución. Además, el concepto de solución de Minimax ha sido descartado por no entrenar correctamente a los agentes. En consecuencia, hemos decidido estudiar el resultado de dos agentes entrenados con Pareto y Nash respectivamente. Usaremos las configuraciones óptimas halladas en el experimento anterior para cada uno de los agentes:

	Agente 0 con Pareto	Agente 1 con Nash
<i>Gamma</i>	0.999	0.99
<i>Epsilon decay</i>	0.7	0.99
<i>Epsilon</i> mínimo	0.3	0.4
<i>Epsilon</i> inicial	1	0.1
<i>Alpha decay</i>	0.95	0.99
<i>Alpha</i> mínimo	0.01	0.01
<i>Alpha</i> inicial	0.01	0.3

En las figuras 38 y 39 se pueden observar las diferentes recompensas para cada uno de los agentes.

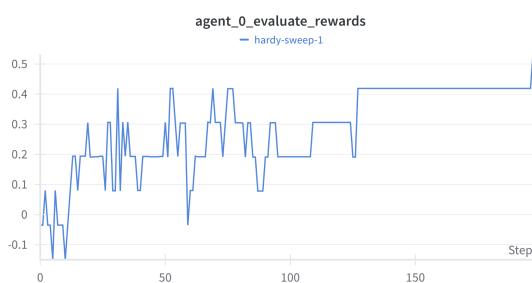


Figura 38: Recompensas para el agente 0 en la evaluación.



Figura 39: Recompensas para el agente 1 en la evaluación.

Vemos como el agente 1 obtiene mejores recompensas a largo plazo comparado con el agente 0. Aunque en el experimento previo concluimos que ambos conceptos de solución eran igual de buenos, ahora Nash está ganando a Pareto. Hay dos posibles explicaciones que justifican este comportamiento

(las dos pueden darse a la vez). La primera es que al usar distintos conceptos de solución, una predomina frente a la otra. Por ejemplo, en el dilema del prisionero, si un agente escogiera seguir como estrategia alguno de los óptimos de Pareto (posiblemente cooperar) mientras que el otro escogiera la estrategia del equilibrio de Nash (siempre traicionar), se vería gravemente perjudicado. La segunda es la diferencia de los valores en los hiperparámetros. La configuración de hiperparámetros fue encontrada cuando ambos agentes tenían la misma configuración. Ahora el agente 0 podría verse perjudicado si la política del agente 1 le permite tomar ventaja al usar una configuración más competitiva.

Finalmente, en las figuras 40 y 41 puede observarse cómo los errores por diferencias temporales muestran patrones similares, por lo que los errores están influenciados por la presencia de otros agentes.

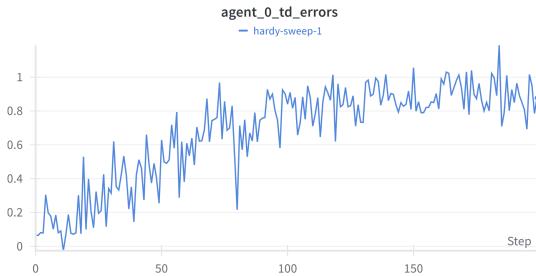


Figura 40: Errores en diferencias temporales para el agente 0 en la evaluación.

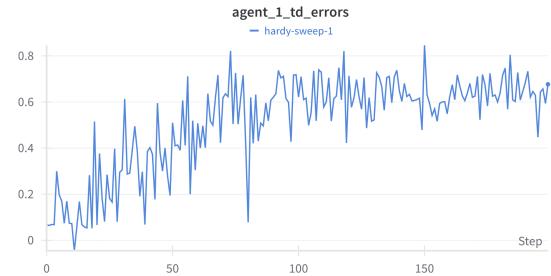


Figura 39: Errores en diferencias temporales para el agente 1 en la evaluación.

## Conclusión

Al entrenar dos agentes con conceptos de solución diferentes, los rendimientos potenciales que puede obtener cada agente puede verse perjudicado. Las recompensas individuales de un agente están influenciadas por el comportamiento del otro agente. A priori parece que los agentes no han conseguido coordinarse, y mientras el agente 1 ha conseguido alcanzar su potencial de recompensa visto en el experimento anterior, la recompensa del agente 0 se ha visto reducida ligeramente respecto al experimento anterior. Como recomendación para entrenar agentes en entornos similares creemos que hay que buscar dos conceptos de solución que casen. Por ejemplo, Minimax y Pareto nunca combinarán ya que los objetivos van en sentido contrario totalmente.

# Experimento 3

En este experimento trataremos de responder la siguiente pregunta:

- ¿Cómo se compara JAL-GT, que es un algoritmo multiagente debido a que calcula la Q en base a acciones conjuntas, con IQL (independent Q Learning)?

Para ello implementaremos el algoritmo IQL, ejecutaremos un experimento de la misma manera que con JAL-GT y compararemos los resultados obtenidos.

Los rangos de los hiperparámetros serán los mismos y los compararemos con el concepto de solución óptima de Pareto.

Bajaremos del mismo modo que en el experimento anterior el tamaño del mapa a 4x4, mantenemos la representación del estado de 1024 bits y el número de agentes a 2. El número de mapas para entrenar seguirán siendo 10 y el número de epochs 200. Para que sean experimentos comparables fijamos la densidad de obstáculos a 0.1, el radio de observación a 1, 10 episodios por epoch y el número de pasos por episodio a 16.

Los rangos a probar serán los mismo que antes, es decir:

- Valores de **gamma** 0.5, 0.95, 0.97, 0.99 y 0.999.
- Para el valor inicial de **epsilon** probaremos los valores 0.4, 0.8, 0.9, 1.0. . Para los valores mínimos de **epsilon** probaremos los valores 0.01, 0.05, 0.1, 0.2, 0.3. Para el **epsilon decay** probaremos los valores 0.7, 0.95, 0.99, 0.999.
- Para el valor inicial de **alpha** probaremos los valores 0.01, 0.05, 0.1, 0.3. Para los valores mínimos de **alpha** probaremos los valores 0.0, 0.0001, 0.001, 0.01. Para el **alpha decay** probaremos los valores 0.7, 0.95, 0.99, 0.999.

Nuestra hipótesis es que el algoritmo JAL-GT, al considerar explícitamente las acciones conjuntas de los agentes en el cálculo de las Q-values, logrará mejores soluciones desde el punto de vista de la eficiencia colectiva, es decir, más cercanas al óptimo de Pareto.

En contraste, esperamos que IQL, al tratar a cada agente de forma independiente, tienda a generar comportamientos más egoístas, en los que la coordinación es subóptima y los agentes no logran evitar colisiones o acciones inútiles con la misma eficacia. Esto debería reflejarse en recompensas medias más bajas o soluciones más dispersas en el espacio de políticas.

Por lo tanto, suponemos que la inclusión explícita de la cooperación (mediante el uso de acciones conjuntas en JAL-GT) será clave para obtener políticas más eficientes en entornos multiagente con intereses parcialmente alineados.

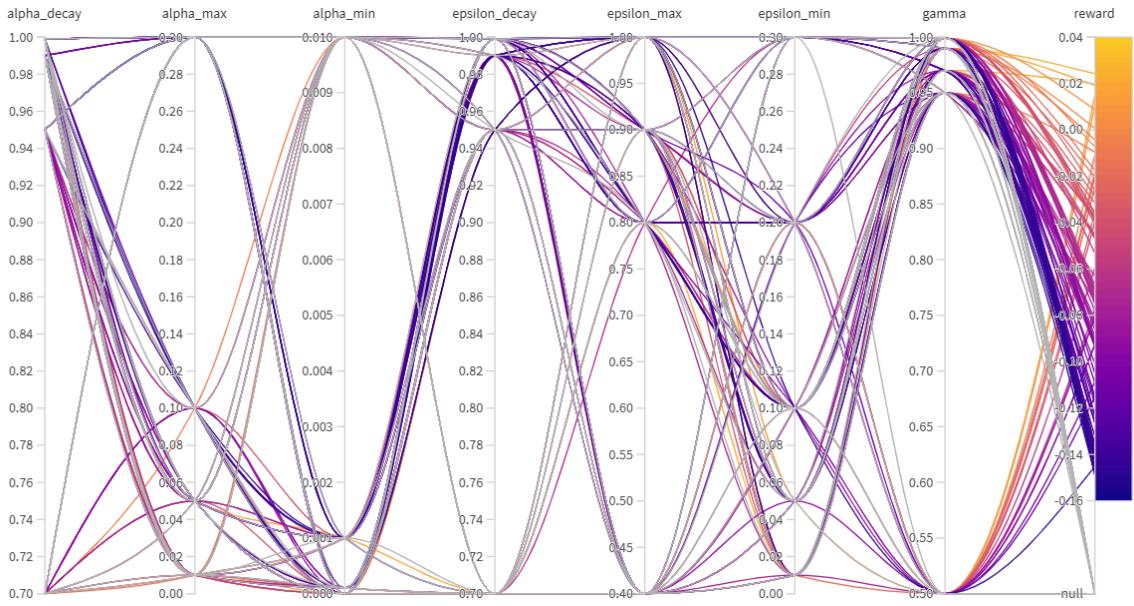


Figura 40: Configuraciones de hiperparámetros en el estudio con IQLearning.

Al igual que en el experimento anterior, usamos Wandb para visualizar la evolución y el impacto de los hiperparámetros sobre la recompensa total obtenida. En la Figura 40 se muestra el gráfico de configuraciones evaluadas con el algoritmo IQL.

Podemos observar que las trayectorias de aprendizaje muestran una mayor variabilidad en las recompensas obtenidas frente a los diferentes valores de hiperparámetros. A diferencia de JAL-GT, donde ciertas combinaciones mostraban una tendencia clara hacia mayores recompensas, en IQL las soluciones parecen más dispersas y menos consistentes, lo cual podría indicar una mayor dificultad para aprender buenas políticas cuando los agentes no coordinan sus decisiones de manera explícita.

Esto refuerza nuestra hipótesis inicial: al no considerar las acciones conjuntas, IQL tiende a converger hacia soluciones menos cooperativas, lo que afecta negativamente al rendimiento global del sistema. A pesar de que algunas configuraciones alcanzan buenos resultados individuales, estas no siempre coinciden con los óptimos de Pareto.

Una posible explicación de estos malos resultados es el concepto de estacionariedad, es decir, el comportamiento egoísta de los agentes hace que frente a un conflicto, los dos tomen la misma acción de resolución que les devuelve a un nuevo estado de conflicto.

Por tanto, concluimos que el uso de soluciones de Pareto permite entender mejor las dinámicas multiagentes en entornos con intereses mixtos. La comparación con JAL-GT sugiere que los algoritmos que modelan explícitamente la interacción entre agentes tienen una mayor capacidad para evitar comportamientos egoístas y encontrar políticas más eficientes desde una perspectiva conjunta.

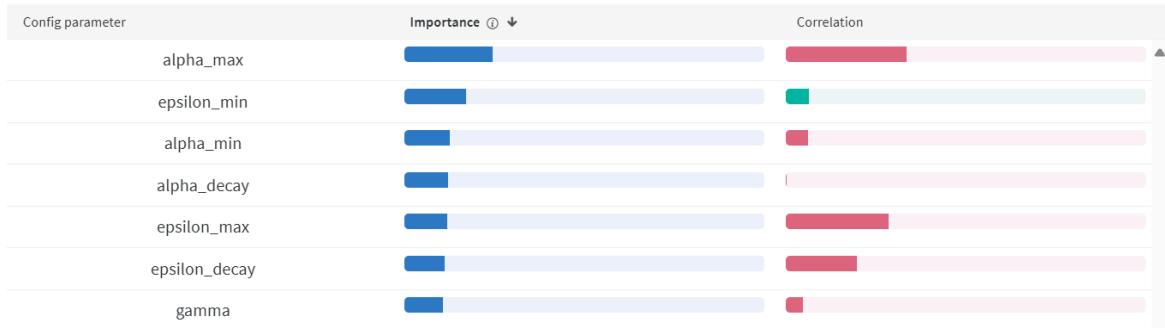


Figura 41: Correlaciones de los hiperparámetros en el estudio.

La Figura 41 nos indica que los parámetros relacionados con la tasa de aprendizaje (**alpha**) y la exploración (**epsilon**) tienen un impacto significativo en el rendimiento del modelo IQL.

Por otro lado, tenemos que tiene una correlación negativa fuerte los parámetros **alpha\_max** y **epsilon\_max**, esto sugiere que valores iniciales demasiado altos de la tasa de aprendizaje o la exploración afectan negativamente el desempeño. El único parámetro con correlación positiva leve es **epsilon\_min** lo que sugiere que mantener un mínimo de exploración ayuda a evitar políticas demasiado prematuras.

Este comportamiento refuerza la hipótesis de que IQL, al carecer de coordinación explícita entre agentes, es muy sensible a la configuración de hiperparámetros. La falta de consistencia en la correlación positiva sugiere que en IQL, encontrar buenas políticas depende más del ajuste fino, mientras que en JAL-GT las soluciones parecen más robustas debido a su diseño colaborativo.

Para estudiar los resultados de la experimentación, nos fijamos en las 10 configuraciones de hiperparámetros que mejores rendimientos han obtenido, como puede verse en la siguiente figura:

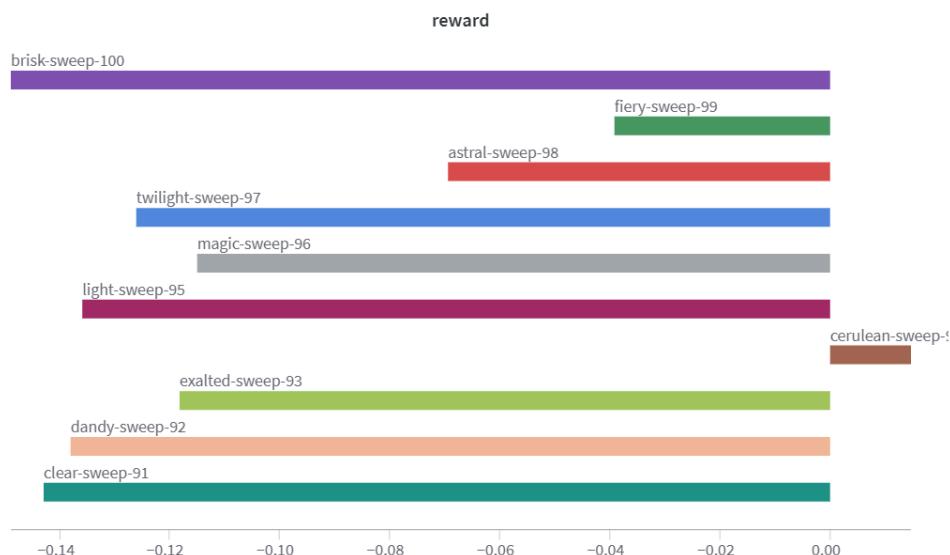


Figura 42: Configuraciones con mejor rendimiento en el estudio de IQLearning.

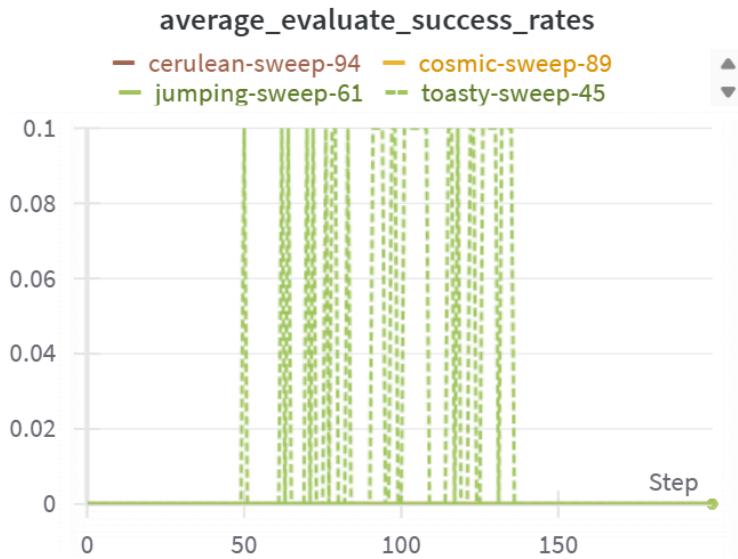


Figura 43: Ratio de éxito en el estudio de IQLearning.

En la Figura 43 anterior podemos observar como las tasas de éxito alcanzadas por estas configuraciones son bajas en términos absolutos, en torno al 0.1 como máximo, lo que indica que incluso las mejores políticas aprendidas por IQL logran resolver solo una fracción muy pequeña de los episodios. Además, las curvas muestran una alta variabilidad y poca estabilidad, con picos esporádicos y caídas bruscas, lo que sugiere que el comportamiento aprendido por los agentes no es robusto ni generalizable.



Figura 44: Diferencias temporales promedio.

Podemos también estudiar el error de diferencias de los algoritmos utilizados por los agentes. En la Figura 44 se puede observar el promedio de suma de errores de los agentes (suma de errores entre número de agentes) mientras que en la Figura 45 y la Figura 46 se pueden observar la suma de errores de los agentes por separado.

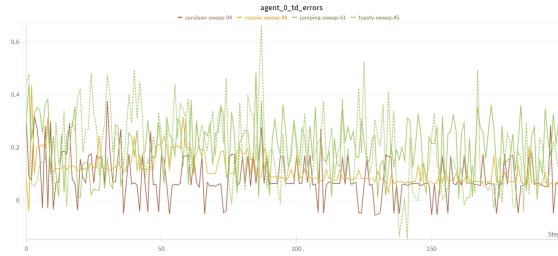


Figura 45: Diferencias temporales para el primer agente.

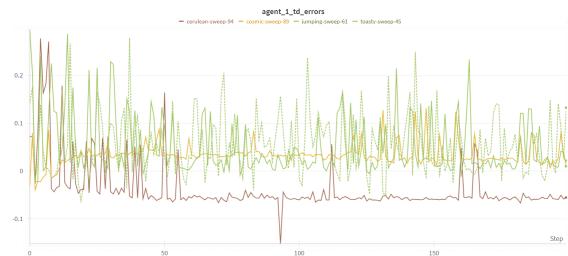


Figura 46: Diferencias temporales para el segundo agente.

Podemos ver como las diferencias temporales promedio parecen seguir diferentes tendencias para diferentes configuraciones de hiperparámetros. Podemos ver como la configuración verde tiene oscilaciones mucho mayores en recompensas generalmente más altas mientras que las otras parecen más estables, sin embargo, no podemos observar una relación clara entre las diferencias temporales y el éxito de los agentes. Al igual que con JAL-GT vemos como las diferencias temporales entre agentes son similares entre ellos.

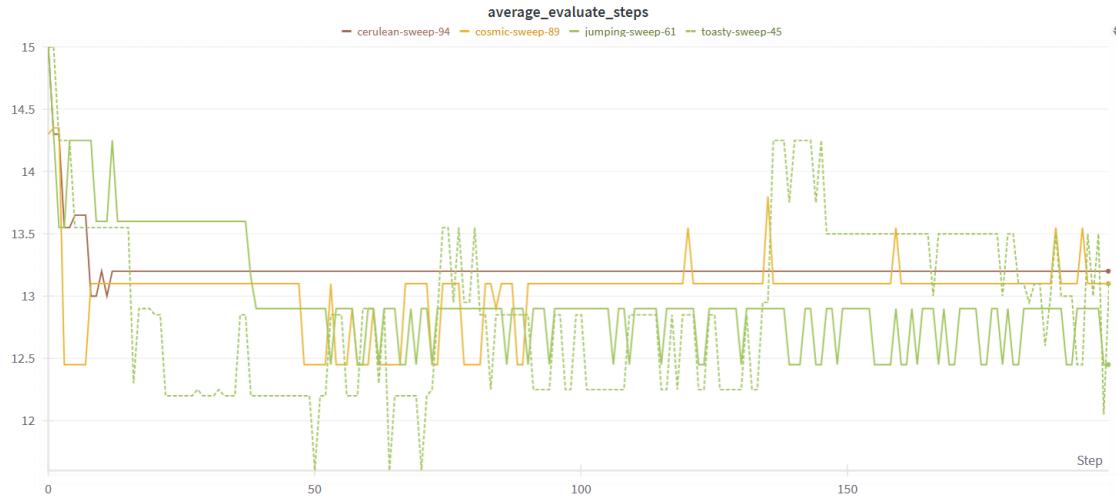


Figura 47: Número promedio de pasos para el estudio de IQLearning.

La evolución de los pasos es la esperada, tiene una tendencia a disminuir respecto a los pasos iniciales, ninguna sorpresa y nada a destacar en la comparación con JALG-AT más allá de que la reducción del número de pasos sea bastante más leve.

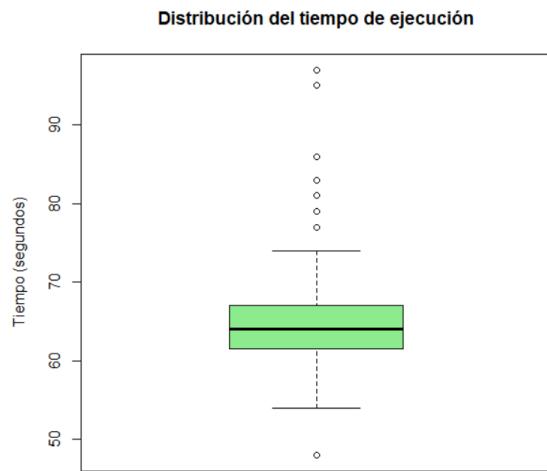


Figura 48: Distribución del tiempo de ejecución para el estudio de IQLearning.

Respecto a la duración de cada ejecución vemos que tienen tiempos muy similares, la duración total del experimento ha sido, igualmente, de 2 horas.

## Conclusión

JAL-GT, al modelar explícitamente la cooperación mediante acciones conjuntas, supera a IQL en aspectos como la estabilidad, el rendimiento y a encontrar soluciones óptimas de Pareto en entornos multiagente con intereses parcialmente alineados.

Por el contrario, IQL muestra un comportamiento más errático y egoísta, generando políticas menos robustas y menos cooperativas. Aunque algunas configuraciones logran buenos resultados puntuales, estas no son representativas, y el éxito global del sistema se ve comprometido por la falta de coordinación.

En consecuencia, para entornos multiagente donde la cooperación y la eficiencia colectiva son cruciales, algoritmos como JAL-GT ofrecen una ventaja significativa frente a enfoques independientes como IQL. Además, el uso del análisis de óptimos de Pareto se consolida como una herramienta valiosa para evaluar soluciones en contextos con múltiples agentes y objetivos parcialmente compartidos.

Para mejorar el rendimiento en entornos multiagente, se recomienda utilizar algoritmos que modelan explícitamente la cooperación, como JAL-GT, especialmente cuando los agentes deben coordinar sus acciones para alcanzar sus objetivos.

Además, incorporar mecanismos de evaluación basados en óptimos de Pareto permite identificar políticas más equilibradas y eficientes, facilitando la exploración de configuraciones que promuevan la colaboración entre agentes.

# Experimento 4

En este experimento trataremos de responder la siguiente pregunta:

- ¿Cómo es capaz de generalizar el algoritmo? Es decir: con el resto de parámetros fijados, ¿cómo es capaz el algoritmo de converger a medida que escala el tamaño del entorno ( $2 \times 2$ ,  $3 \times 3$ ,  $4 \times 4$ ,  $8 \times 8$ , etc), el número de agentes (2, 3, 4, etc), la complejidad de la representación del estado, etc.?

Para este experimento suponemos que la generalización del algoritmo es muy mala ya que un aumento de una unidad en el número de agentes se ve reflejado de forma exponencial en el espacio total.

Fijamos los parámetros en los óptimos encontrados en el primer experimento para la solución de óptimo de Pareto. Es decir, fijamos los valores:

- Alpha Decay = 0.95
- Alpha Inicial = 0.01
- Alpha mínimo = 0.01
- Epsilon Decay = 0.7
- Epsilon Inicial = 1
- Epsilon Mínimo = 0.3
- Gama = 0.999

Para poder definir la convergencia haremos una comparación con los tiempos de ejecución del experimento 1. Es decir, algo más de 65 segundos por ejecución.

Con los parámetros fijados primero probaremos de usar 3 agentes y un mapa  $4 \times 4$ . Luego, probaremos usar 2 agentes y un mapa de  $8 \times 8$ .

## Tres agentes

Sabemos que aumentar el número de agentes, teóricamente, tiene un efecto exponencial en el tiempo de ejecución del algoritmo. Suponemos que del mismo modo será así en la práctica.

Una vez lanzado el experimento con tres agentes, el tiempo de ejecución total ha sido de 47 minutos. La comparación con los tiempos de antes es mayor de lo que esperábamos. Tarda unas 40 veces más en ejecutar un experimento, lo que nos indica una mala generalización. Si en vez de haber fijado unos parámetros hubiéramos utilizado Wandb para encontrar alguna configuración óptima, el problema ya habría sido intratable.

## Mapa de $8 \times 8$

Por otro lado, aumentamos el número de casillas, en este caso podemos suponer que el tiempo de ejecución aumentará bastante ya que las casillas se multiplican por 2. Una vez terminada la ejecución

del experimento, el tiempo de ejecución obtenido es de 46 minutos, esto nos indica que también obtenemos una mala generalización ya que el tiempo es 10 veces mayor de lo esperado, aumentar el tamaño del mapa tiene una gran repercusión en el tiempo del experimento. Por eso, concluimos que de la misma forma que el número de agentes, aumentar el tamaño del mapa hace que el problema sea intratable.

## Conclusión

El experimento confirma que el algoritmo presenta una mala capacidad de generalización cuando se incrementa el número de agentes o el tamaño del entorno. Ambos factores provocan un crecimiento exponencial del tiempo de ejecución, lo que se traduce en tiempos de ejecución excesivos y dificultad para converger pese a usar hiperparámetros que habíamos verificado como óptimos

Para entrenar agentes en entornos que escalan en complejidad, se pueden utilizar representaciones del estado más compactas y aplicar técnicas de poda o aproximación de valor para reducir la complejidad. Además, se puede considerar el uso de algoritmos centralizados o basados en aprendizaje profundo para mejorar la generalización y la eficiencia computacional en escenarios de mayor escala.

# Experimento 5

En este experimento responderemos a la pregunta:

- ¿Es capaz el algoritmo de generalizar de manera que tenga éxito evaluando mapas que no ha visto en entrenamiento?

La hipótesis inicial es que si en el experimento hay un reward alto, este descenderá para los mapas no vistos, sin embargo, no esperamos que decaiga demasiado ya que el agente ha tenido un aprendizaje, y en un mapa tan pequeño debería tener éxito aunque este sea menos frecuente.

En este caso fijamos los parámetros con los óptimos calculados en el experimento 1 para el concepto de solución de Pareto. Después de ejecutar el entrenamiento y evaluarlo con los mapas entrenados haremos 10 evaluaciones con mapas no utilizados previamente.

Los resultados con los mapas vistos en el entrenamiento son de un *reward* medio de 0.6975 y el *success rate* es de 0,6. Por otro lado cuando probamos de evaluar mapas nuevos observamos que el reward medio pasa a ser de 0.36 y el success rate de 0.3. Esto nos confirma la hipótesis, mantiene un reward bastante por encima del 0 y el success rate desciende en un 50% pero se mantiene aceptable.

## Conclusión

El algoritmo muestra una capacidad limitada de generalización a mapas no vistos, con una caída significativa pero no catastrófica en rendimiento. Aunque el éxito disminuye, el agente conserva cierto aprendizaje transferible.

Para mejorar la generalización, se puede incluir una mayor diversidad de mapas durante el entrenamiento o emplear técnicas como **data augmentation** o **entrenamientos con entornos aleatorizados**.

## Extra:

### Introducción y objetivos:

Para este experimento explicaremos cómo funciona el algoritmo REINFORCE para entrenar agentes en el entorno multiagente POGEMA. REINFORCE es un algoritmo de aprendizaje por refuerzo *single-agent*, por ello el enfoque que escogemos será el entrenamiento en paralelo de todos los agentes con un REINFORCE independiente para cada uno. Para usar REINFORCE en un entorno *multi-agent* se hace la asunción implícita de que el resto de agentes en el entorno se pueden modelar como parte de este a efectos prácticos. De este modo el agente se entrena asumiendo que está solo, pese a que no lo esté.

Este experimento plantea distintas preguntas, muchas de ellas similares a las que también plantea la experimentación con cada agente siendo entrenado con un algoritmo Q-Learning, que es también *single-agent*. La primera de ellas es si con este entorno funciona mejor un algoritmo que no sea *multi-agent*.

Otra pregunta que nos surge es si REINFORCE, es el algoritmo adecuado para este tipo de entorno. Esta pregunta está principalmente motivada por la práctica 2, en la cual, a pesar de probar con muchas combinaciones muy variadas de parámetros, no llegamos a encontrar ninguna combinación de parámetros mínimamente buena. Este entorno y Cliff-Walking se parecen en ser un pequeño mapa en el cuál los agentes deben llegar a una casilla objetivo. Por otro lado en este a diferencia de Cliff-Walking tenemos que hay más de un agente, no hay aleatoriedad en el resultado de las acciones. Por último, el cambio más importante es la ausencia de casillas trampa con reward muy negativo y la existencia de una recompensa negativa para cada casilla a la que nos movemos distinta a la del objetivo. Finalmente, cabe destacar que puede considerarse que la preferencia de los agentes con índice menor para moverse a una casilla en caso de conflicto puede guardar cierta equivalencia con el *slippery* de Cliff-Walking en cuanto a los efectos para un agente. De todos modos, pensamos que esta equivalencia es muy limitada por dos motivos: el primero que, probablemente, es mucho más fácil de modelar la colisión de movimientos de dos jugadores que el efecto del *slippery*, y el segundo, que su prevalencia sobre el total de movimientos es menor.

### Hipótesis:

Para este experimento planteamos las siguientes hipótesis:

En primer lugar, pensamos que, pese al hecho de que estamos en entorno multi-agente, es posible que los agentes rindan bien con un algoritmo *single-agent*. Esto es porque partiendo de la definición de entorno multi-agente (esto es: “el entorno es multiagente si hay como mínimo: un agente A, y un agente **relevante** para A”) y tomando el ser multi-agente como una gradación y no como un atributo booleano, nos damos cuenta de que este no es un entorno *muy multi-agente*. Esto es porque todos los agentes tienen sus objetivos independientes y el impacto que tienen los unos sobre los otros (fruto de la colisión) no es excesivamente grande por no ser muy común y por ser poco perjudicial en cuánto a recompensa.

En segundo lugar, pensamos que REINFORCE podría ofrecer un mejor rendimiento en este caso que en el entorno Cliff-Walking. Esto se debe a que no hay una recompensa exageradamente negativa para una casilla que pueda contaminar toda una trayectoria, dificultando mucho el discriminar qué acciones son realmente malas y cuáles pese a ser buenas se hallan en una trayectoria con una acción mala.

Por otro lado, al tener en cuenta en cada episodio, el conjunto de acciones del episodio para aprender, REINFORCE puede ser un algoritmo más robusto y con mejor rendimiento que IQL, que solo tiene en cuenta pares estado-acción y su recompensa. Esta última diferencia podría llevar a que REINFORCE sortee mejor problemas como la estacionariedad del entrenamiento.

Con respecto a los parámetros de REINFORCE hipotetizamos que no serán necesarios learning\_rates y factores de descuento tan bajos como para Cliff-Walking. Esto se debe a que, en primer lugar, el espacio de estados es menor en los mapas que probamos. Por otro lado, el hecho que el estado se derive de las observaciones y que probemos distintos mapas (lo que a su vez implica mayor combinación de puntos en los que puede estar el objetivo del resto de agentes respecto al nuestro) lleva a que *naturalmente* el agente se vea expuesto a una mayor diversidad de estados, es decir que *por defecto* el entorno lo lleve a explorar más. Respecto al *learning rate*, mantenemos la hipótesis de que debe ser alto para no dejar de aprender muy rápido.

## Diseño de los experimentos:

El diseño de este experimento es sencillo y se parece al de experimentos anteriores. En primer lugar, tendremos una serie de parámetros fijos, luego usando wandb trataremos de ajustar los valores de los otros parámetros hasta encontrar aquellos que dan el mejor reward.

### Parámetros fijos:

Estos parámetros están fijos en toda la experimentación, entre paréntesis ponemos su nombre en el código de la clase *Parameters*:

- **Densidad de obstáculos** (obstacle\_density): La probabilidad de tener un obstáculo en el mapa, fijada en 0.1.
- **Número de agentes** (num\_agents): La cantidad de agentes en el mapa, establecida en 2.
- **Tamaño del mapa** (map\_size): Las dimensiones del mapa, configuradas a 4 (lo que implica un mapa de 4×4).
- **Número de mapas** (num\_maps): La cantidad de mapas para entrenar y evaluar, fijada en 10 (se repiten si el número de episodios es mayor que el de mapas).
- **Épocas** (epochs): El número total de épocas, donde cada una incluye un entrenamiento y una evaluación, fijado en 200.
- **Episodios por época** (episodes\_per\_epoch): El número mínimo de episodios por época de entrenamiento, establecido en 10.
- **Pasos máximos por episodio** (max\_episode\_steps): El número máximo de pasos permitidos por episodio, fijado en 16 (el episodio se trunca si se excede).
- **Radio de observación** (observation\_radius): El radio de visión de los agentes, con un valor de 1.

## Parámetros que buscamos optimizar:

Los siguientes parámetros son los que tratamos de ajustar para encontrar el *reward* óptimo. Entre paréntesis ponemos el nombre de los parámetros en *Parameters* y en el archivo *json* con los parámetros de *sweep* que pasamos a *wandb*.

- **Learning rate** (alpha\_max): probaremos los valores 0.99, 0.95, 0.9, 0.75, 0.5, 0.35 de modo que cubrimos un rango de valores entre 0 y 1 muy amplio, aunque probamos más en la franja superior (dado que creemos que el óptimo estará ahí, si no se halla allí experimentamos más en valores inferiores). Esto nos permitirá ajustar valores y a la vez tener una visión completa del comportamiento del algoritmo en función del *learning rate*.
- **Learning rate decay** (alpha\_decay): probaremos los valores 0.999, 0.99, 0.9, no probamos valores por debajo de 0.9 porque se traducen en un *learning rate* nulo tras los primeros episodios. Con esto tendremos una visión completa de los valores de *learning rate* con sentido.
- **Factor de descuento** (gamma): explicaremos los siguientes valores 0.99, 0.95, 0.9, 0.75, 0.5, 0.35 y 0.1. De nuevo, cubrimos un rango de valores entre 0 y 1 muy amplio probando más entre los cercanos a uno para ajustar ya directamente si se cumple la hipótesis con respecto al factor de descuento. Tenemos con estos valores una visión completa y precisa del *reward* obtenido en función de los distintos parámetros.

## Uso de *wandb*:

Con *wandb* ejecutaremos 126 *sweeps* en este espacio de posibles configuraciones de modo que podemos llegar a cubrir todo el espacio ( $6 \cdot 3 \cdot 7 = 126$ ). Luego extraemos los resultados de *wandb* y los estudiaremos.

## Resultados:

El experimento ha arrojado muy buenos resultados, hasta el punto que nos ha sorprendido positivamente. El máximo reward medio alcanzado ha sido 0.84852, hallado con la configuración de parámetros learning rate 0.95, learning rate decay 0.999 y factor de descuento 0.99. En todos los subóptimos siguientes el decay es 0.999 como se había hipotetizado al inicio. El factor de descuento y el learning rate son incluso un poco más altos de lo esperado, pero parecen confirmar la hipótesis en cuanto a ser altos. Con respecto a esto hay un fenómeno merecedor de atención en futuros experimentos, el segundo subóptimo mantiene todos los parámetros excepto que baja el learning rate a 0.9, pero en el tercero se mantienen todos los parámetros excepto el learning rate que es de 0.35. Esto se podría deber a que tanto para un agente que sigue el camino de la exploración como para uno que sigue el de la explotación se acaba llegando al óptimo de manera que este parámetro tiene varios valores que permiten llegar al óptimo.

Fijémonos ahora en los resultados del conjunto de *sweeps* arrojados por *wandb*:

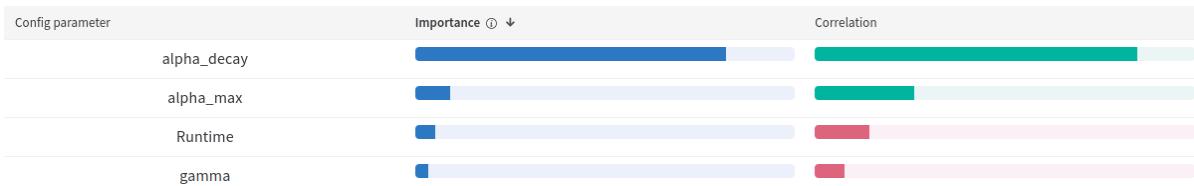


Figura 49: Importancia y correlación de los distintos parámetros con respecto al *reward* medio de los agentes.

Como se ha dicho ya la importancia de un learning rate decay muy alto es capital, por otro lado, también vemos una correlación positiva más débil con el learning rate como se había comentado. Por último el factor de descuento tiene una ligera correlación negativa y baja importancia, en la línea de lo comentado antes. Este último fenómeno se deja para trabajos posteriores por tener una importancia baja y requerir mucha experimentación nueva, lo más probable es que sencillamente el factor de descuento tenga poco impacto y la ligera correlación negativa se deba a la elección de parámetros.

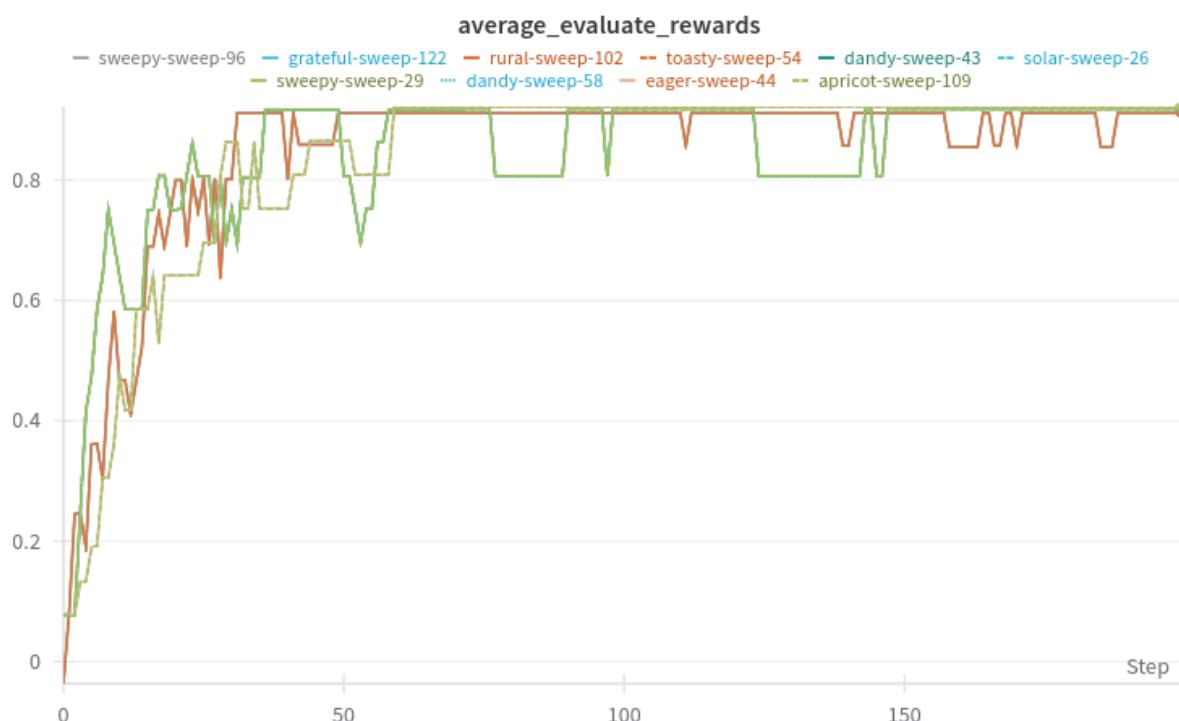


Figura 50: reward medio por epoca con las mejores configuraciones.

Si miramos la evolución del *reward* medio a lo largo de la experimentación, vemos que para distintas configuraciones (de entre las mejores) se converge rápidamente a la política óptima alrededor del episodio 50.

## Conclusiones y recomendaciones para futuros experimentos:

Vemos pues, que este algoritmo es ideal para un entorno como el que estamos tratando. En concreto arroja mucho mejores resultados que los otros al ofrecer recompensas sustancialmente mayores y una convergencia mucho más rápida. Los resultados parecen confirmar que REINFORCE es bueno para

evitar la estacionariedad y que es un algoritmo que puede dar buenos resultados al ser ejecutado en paralelo pero independientemente para entrenar agentes en un entorno multiagente.

De cara a su uso futuro en entornos similares, recomendamos usar un learning rate decay muy cercano a 1 (0.999 por ejemplo), un learning rate un poco menor pero también alto (entre 0.9 - 0.95). Finalmente, en cuanto al factor de descuento recomendamos probar un rango de valores amplio y luego hacer una segunda vuelta alrededor del mejor ya que, el experimento no arroja resultados claros en este sentido.

## Conclusiones

En esta práctica hemos aprendido los diferentes enfoques que permiten abordar un entorno multiagente usando entrenamiento por refuerzo. Hemos visto cómo el algoritmo de JAL-GT permite que los agentes encuentren de forma eficiente las políticas óptimas implementando con el concepto de solución que más se adecue en cada caso. Por otro lado, el algoritmo IQL no presenta buenos resultados en este entorno multiagente. Por otro lado, hemos visto las dificultades que tiene el algoritmo JAL-GT para escalar a medida que aumenta el número de agentes o el tamaño del entorno. Esta limitación ha impedido hacer un estudio que fuese más general, quedando el estudio a dos agentes y tamaño de mapa de 4x4. El algoritmo de REINFORCE presenta una mejora significativa obteniendo resultados mejores que los experimentos usando JAL-GT. REINFORCE evita la estacionariedad y converge más rápido a la política óptima obteniendo además recompensas mejores.

Como posibles mejoras y extensiones, consideramos la posible implementación de un histórico de observaciones como estado, en vez de la codificación a mano de estados de 10 bits. También, es interesante explorar implementaciones que utilizan redes neuronales o representaciones vectoriales como estado, lo cual podría salvar la no tratabilidad en cuanto el tamaño del problema aumenta.