Lab 2

Sistemas Inteligentes Distribuidos

TAXI-V3 GYMNASIUM AGENT

Momin Miah Begum
Rubén Catalán Rua
Jianing Xu
Muhammad Yasin Khokhar

QP: Mayo 2024

Índice

1. Introducción

En esta práctica, se evaluarán y compararán varios algoritmos de aprendizaje por refuerzo. Estos son: Iteración por Valor, Estimación Directa Basada en un Modelo y Q-learning. Para ello, usaremos el entorno *Taxi-v3* que nos facilita Gymnasium.

program_name = sys.argv[1] parameter_name = sys.argv[2]

El espacio de acciones es discreto y contiene 6 acciones posibles:

- 0: Moverse hacia el sur (abajo).
- 1: Moverse hacia el norte (arriba).
- 2: Moverse hacia el este (derecha).
- 3: Moverse hacia el oeste (izquierda).
- 4: Recoger al pasajero.
- 5: Dejar al pasajero.

El espacio de observación también es discreto y contiene 500 estados. Estos estados se derivan de las combinaciones posibles de 25 posiciones del taxi, 5 posibles ubicaciones del pasajero (incluyendo cuando el pasajero está en el taxi) y 4 ubicaciones de destino.

El agente tiene como objetivo maximizar las recompensas acumuladas durante un episodio. Las recompensas se distribuyen de la siguiente manera:

- +20 por dejar al pasajero en el destino correcto.
- -10 por intentar recoger o dejar al pasajero ilegalmente.
- -1 por cada paso dado que no resulte en otra recompensa.

El episodio termina en las siguientes condiciones:

- Terminación: El taxi deja al pasajero en el destino correcto.
- Truncamiento: La duración del episodio alcanza el límite de 200 pasos.

Nos proporcionan funciones útiles como *step()* y *reset()* que retornan un diccionario con las probabilidades y si la accion nos llevara a otro estado

En este documento presentamos, además de una breve explicación de los conceptos más importantes de cada algoritmo implementado, una serie de experimentos para poder ver en qué medida afectan (positiva o negativamente) ciertos parámetros respecto a cada algoritmo. Finalmente, haremos una breve comparación entre los algoritmos.

2. Iteración por Valor

La iteración por valor es un algoritmo que trabaja sin conocimiento de la política del agente. Basándose en una señal de recompensa y un entorno observable, con una $\gamma < 1$, podemos encontrar la política óptima del agente.

2.1. Implementación

Para implementar un agente entrenado por iteración de valor, creamos una clase *ValueIterationAgent*, que tiene como parámetros el entorno escogido (en esta práctica: Taxi-v3), un array inicializado a 0 que representa el valor inicial de todo el espacio de estados del entorno, y el factor de descuento utilizado, que nos indica cuánto valoramos las recompensas futuras.

Clase ValueIterationAgent

- **calc_action_value(state, action)**: Calcula la recompensa esperada de tomar una acción en un estado dado.
- **select_action(state)**: Selecciona la acción con la máxima recompensa esperada para un estado dado.
- **value_iteration()**: Realiza una iteración del algoritmo de iteración de valor.
- **policy()**: Devuelve la política del agente tras el entrenamiento.

Funciones adicionales

- **check_improvements(agent, env, num_episodes, t_max)**: Calcula la recompensa media obtenida tras *num_episodes* iteraciones del agente en su estado de entrenamiento actual.
- **train(agent, env, reward_threshold, num_episodes, t_max)**: Entrena al agente hasta que la recompensa obtenida supere un valor *reward_threshold*.

El *main* del programa recoge el parámetro que se va a probar y los valores a utilizar. Posteriormente, se realiza un entrenamiento del agente con cada uno de estos valores y un número *num_test_episodes* de pruebas sobre las cuales recogemos los resultados y calculamos las medias de recompensas obtenidas.

2.2. Experimentación y Resultados

Los dos parámetros que se probarán serán el factor de descuento y el umbral de recompensa. Cuando uno de estos se esté modificando, el otro se mantendrá constante con un parámetro predeterminado.

Parámetros base:

- GAMMA: 0.9
- REWARD_THRESHOLD: 3

Durante el entrenamiento del agente, se calculará el tiempo medio requerido por cada episodio, así como el total de tiempo del entrenamiento. Se obtendrán los valores de recompensa obtenidos hasta superar el umbral y se generarán tablas para comparar los valores de tiempo y recompensa entre los valores de prueba del parámetro.

Pruebas con gamma

Hipótesis inicial: Debido a que estamos en un entorno relativamente pequeño, las recompensas futuras nos interesarán gratamente. Se esperan mejores resultados con gammas más cercanas a 1.

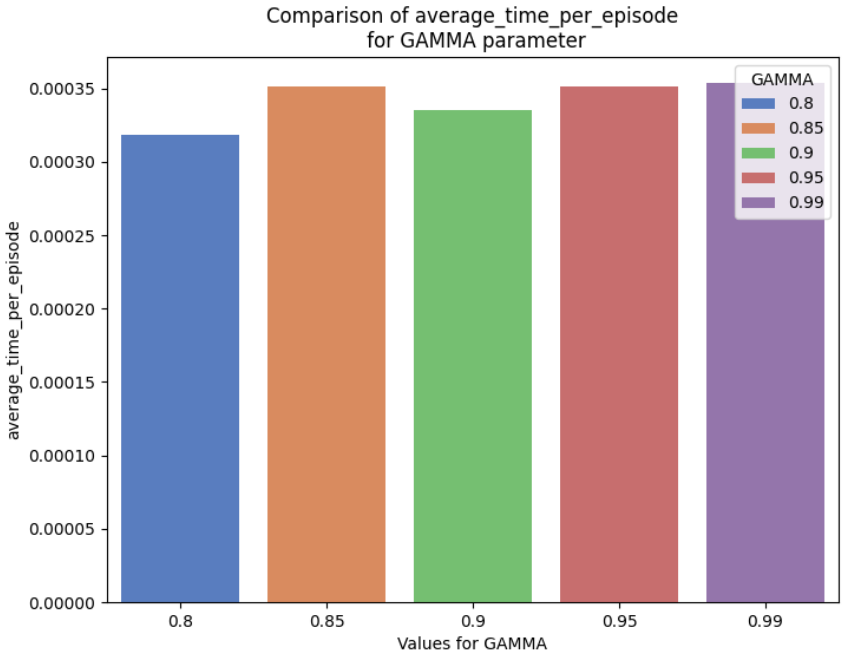

Figura 1: Tiempo medio por episodio

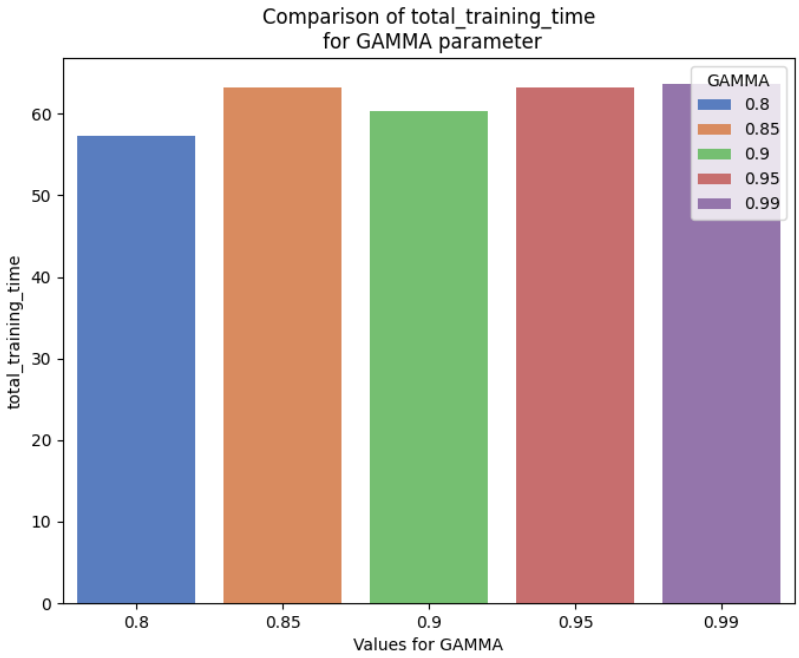

Figura 2: Tiempo total de entrenamiento

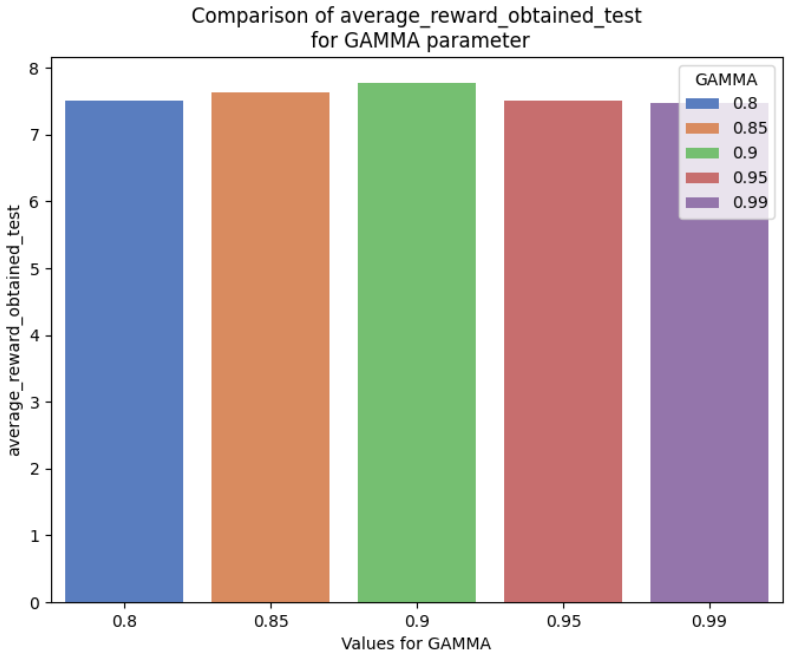

Figura 3: Recompensas medias obtenidas tras el entrenamiento

Con estos resultados, vemos que la máxima recompensa se obtiene cuando nuestro parámetro GAMMA = 0.9, con un valor de **7.775**, mientras que el entrenamiento más rápido se obtuvo con GAMMA = 0.8. Hay un argumento para escoger cada una de estas opciones, ya que las diferencias de tiempo y recompensa obtenidas es mínimo. Si el tiempo apremia, quizá una GAMMA más baja sea la opción más interesante, pero en caso contrario, que asumimos es el más común, GAMMA = 0.9 obtendrá los mejores resultados.

Pruebas con REWARD_THRESHOLD

Habiendo terminado las pruebas con la gamma, pasamos a probar el umbral de recompensa. Es decir, el valor a superar antes de terminar el entrenamiento del agente.

Hipótesis inicial: Un umbral más alto provocará más iteraciones del algoritmo, buscando políticas más beneficiosas. Es natural asumir que se conseguirán mejores recompensas cuanto más alto sea.

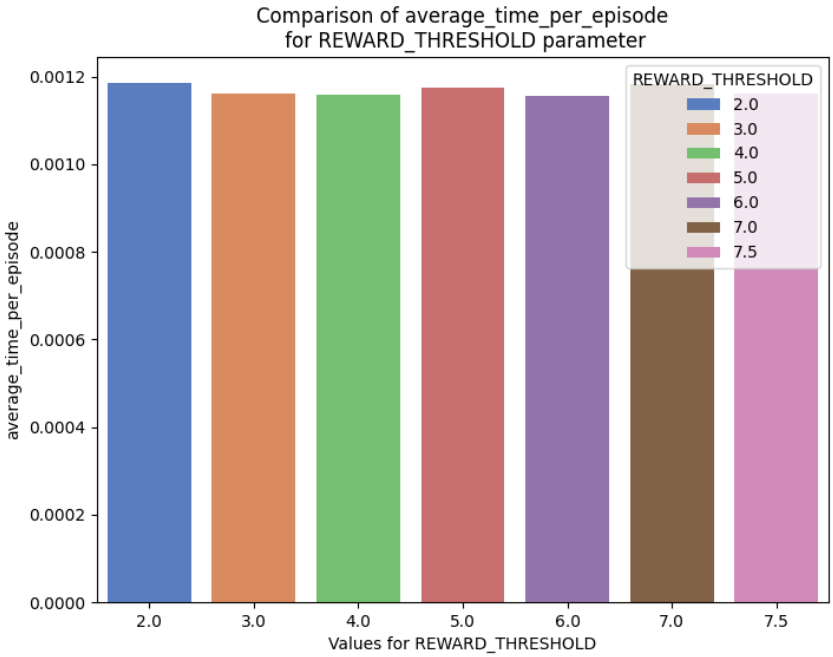

Figura 4: Tiempo medio por episodio

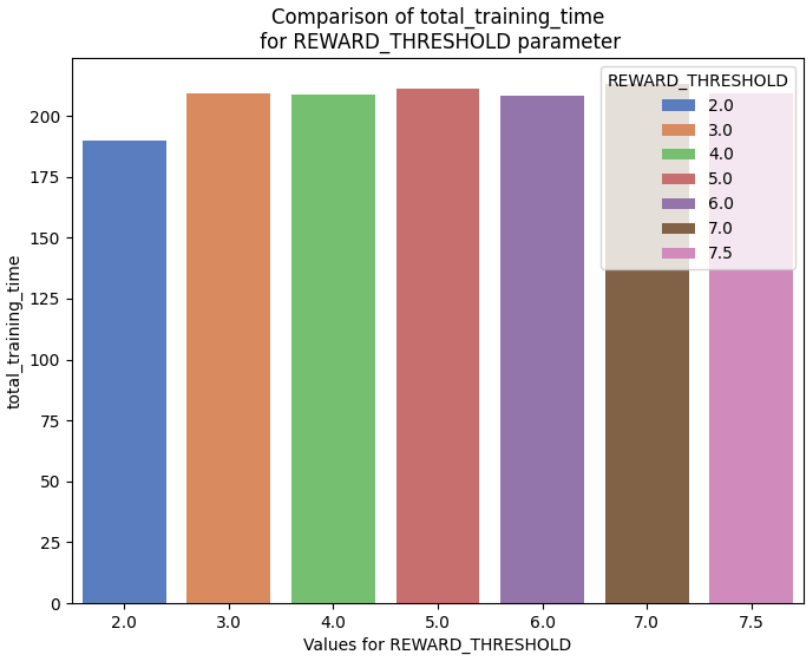

Figura 5: Tiempo total de entrenamiento

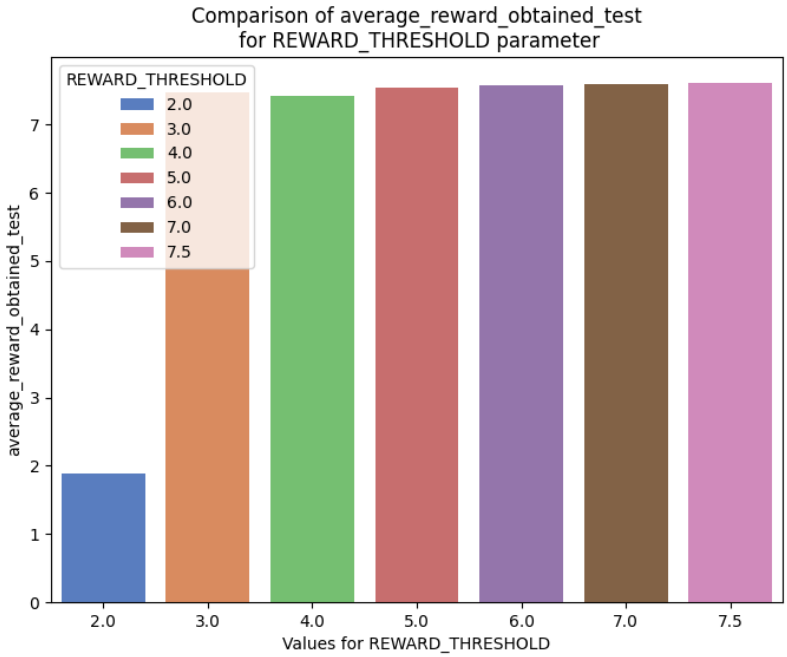

Figura 6: Recompensas medias obtenidas tras el entrenamiento

Vemos que, si bien el promedio en el tiempo de los episodios es mayor con un umbral de 2, el tiempo total es inferior, esto se debe a que para todos los otros umbrales, se realiza una ejecución más del algoritmo para que la recompensa obtenida la supere, que es la razón por la extrema diferencia en el promedio de recompensas entre REWARD_THRESHOLD = 2 y REWARD_THRESHOLD = 3

Se observa que la recompensa máxima se consigue con un umbral de 7.5, siendo el valor **7.611**, si bien todas las recompensas medias a partir de un umbral >= 5 son superiores a 7.5. Consideramos que es correcto mantener un umbral más cercano o igual a 7.5 para evitar recompensas inferiores debido a variancias en las ejecuciones. [1].

Conclusión

Vemos que en base a estos resultados, el mejor entrenamiento se conseguirá con una GAMMA = 0.9 y un THRESHOLD de 7.5. Que el umbral de 8 generará una ejecución infinita nos hace llevar a creer que este es un valor de recompensa no alcanzable dentro del entorno sobre el cual trabajamos.

3. Model-Based Estimación Directa

La Estimación Directa Basada en Modelo es un algoritmo simple y ciertamente ineficiente el cual combina iteracion por valor a un modelo aproximado el cual ha sido determinado a partir acciones aleatorias y recogiendo (experiencia) datos del entorno.

3.1. Implementación

El algoritmo para entrenar al agente y obtener la política óptima consta de varias partes diferenciadas:

Primero, se recopilan datos para aproximar el modelo. Para aprender las transiciones y recompensas del entorno, se recopilan datos a partir de una serie de movimientos aleatorios. Esto se realiza mediante la función `play_n_random_steps`, que actualiza las tablas `rewards` y `transits`. La tabla `rewards` almacena las recompensas asociadas con cada transición de estado y acción, mientras que `transits` cuenta cuántas veces una acción en un estado específico lleva a otro estado.

[1] No se han mostrado pruebas con THRESHOLD = 8 ya que no terminaba la ejecución

El vector `rewards` se utiliza posteriormente para calcular el valor de una acción en un estado, lo cual es esencial para elegir la mejor acción. El vector `transits` se emplea para calcular la probabilidad de llegar a un estado determinado dado un estado y una acción específicos. Al finalizar esta fase, obtenemos una aproximación de las recompensas asociadas con cada acción y estado, construyendo así un modelo del entorno basado en la experiencia adquirida mediante movimientos aleatorios.

En la segunda parte, se realiza la iteración por valor para actualizar la tabla de valores `V` para cada estado, calculando una lista con los valores de cada acción posible mediante la función `calc_action_value`.

A continuación, se detallan las principales funciones del agente y su propósito:

- `play_n_random_steps(count)`: Realiza una serie de movimientos aleatorios para recopilar datos sobre las transiciones y recompensas del entorno.
- `calc_action_value(state, action)`: Calcula el valor de una acción en un estado dado, utilizando las recompensas y las probabilidades de transición almacenadas.
- `select_action(state)`: Selecciona la mejor acción posible en un estado dado, basándose en los valores calculados.
- `value_iteration()`: Realiza la iteración por valor para actualizar la tabla de valores `V` hasta que se cumpla la condición de convergencia.
- `policy()`: Genera la política óptima basada en los valores calculados para cada estado y acción.
- `test_episode()`: Ejecuta un episodio de prueba para evaluar el rendimiento de la política aprendida.

Este enfoque permite construir un modelo del entorno basado en la experiencia adquirida mediante movimientos aleatorios y utilizar dicho modelo para aprender una política óptima mediante iteración por valor.

3.2. Experimentación y resultados

Para evaluar cómo influyen los parámetros sobre la calidad del agente, procederemos de la manera más sencilla: fijaremos todos los valores excepto el que vayamos a observar. Probaremos rangos de valores que consideremos adecuados y confirmaremos o refutaremos nuestras hipótesis.

Los parámetros que tendremos en consideración son:

1. **Factor de Descuento (GAMMA)**: Este parámetro determina cuánto valor le damos a las recompensas futuras en comparación con las inmediatas.

- **Hipótesis**: Creemos que un valor alto de GAMMA (cercano a 1) permitirá al agente considerar más las recompensas futuras, lo cual debería mejorar el rendimiento general del agente al hacer que planifique a largo plazo.

Vamos a experimentar con el agente utilizando los siguientes valores de gamma: 0.6, 0.75, 0.95 y 0.99. El motivo por el cual hemos escogido estos valores es que nos permiten analizar casos extremos, como 0.6 y 0.99. Estos valores nos permitirán comprender cómo el agente se desarrolla al dar mucha prioridad a las recompensas futuras o poca relevancia a estas. Además, para tener una idea completa o un estudio continuo, incluimos los valores 0.75 y, por defecto, el 0.95. Por otra parte, la resta de parámetros son;

1. T_MAX = 25: Pensamos que este número de pasos es más que suficiente para este entorno.
2. REWARD_THRESHOLD = 0.9: Mantenemos el valor por defecto.
3. NUM_TRAJECTORIES = 1000: Queremos asegurarnos de que el agente prepare bien el modelo, por eso realizamos un número elevado de pasos aleatorios al principio.

A continuación, presentamos los gráficos obtenidos al poner a prueba el agente en este entorno. Los otros parámetros del modelo no se modifican; se pueden consultar en el código adjunto a la práctica.

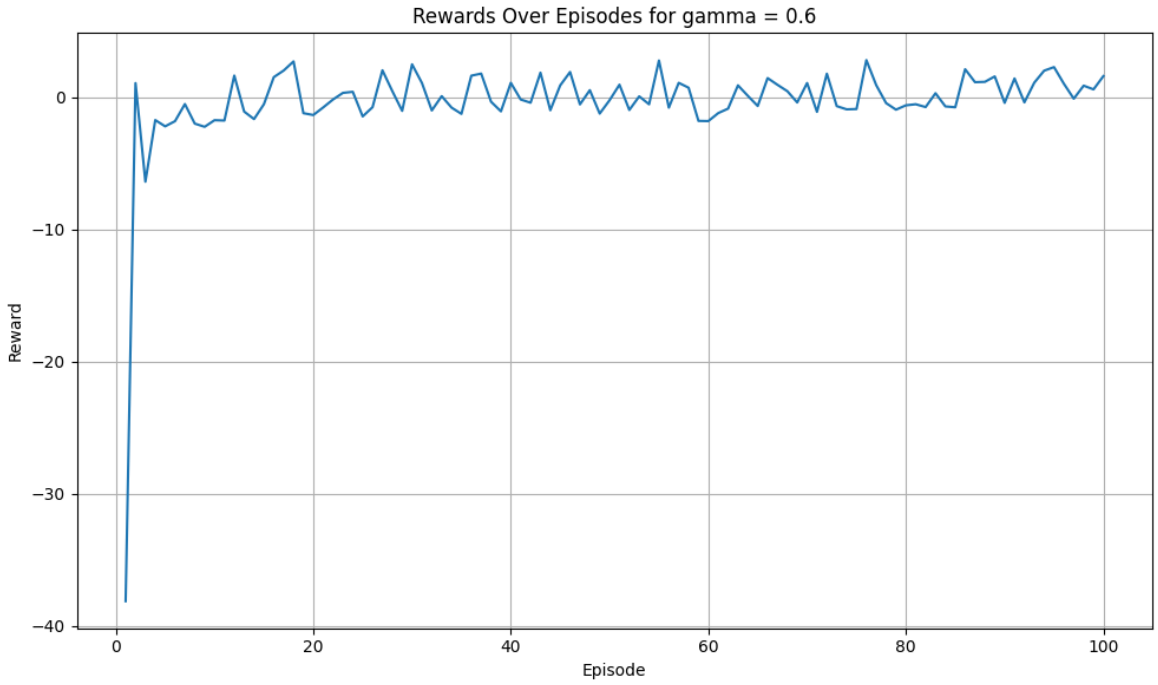

Figura 7: Recompensas para $\gamma = 0{,}6$

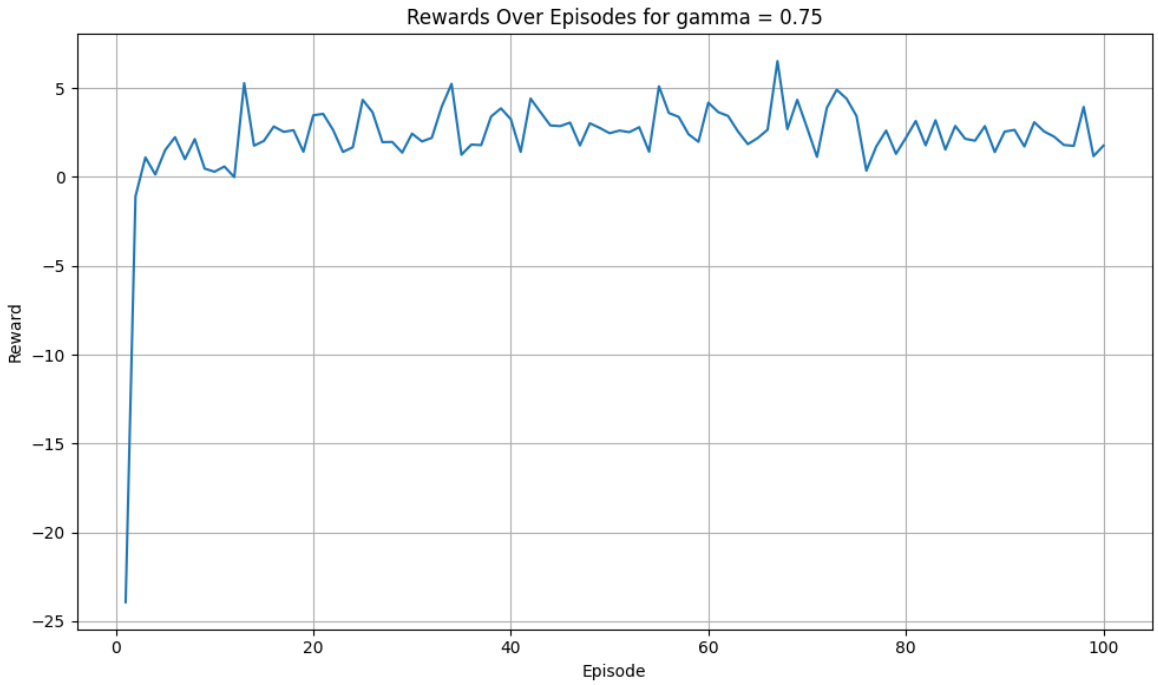

Figura 8: Recompensas para $\gamma = 0{,}75$

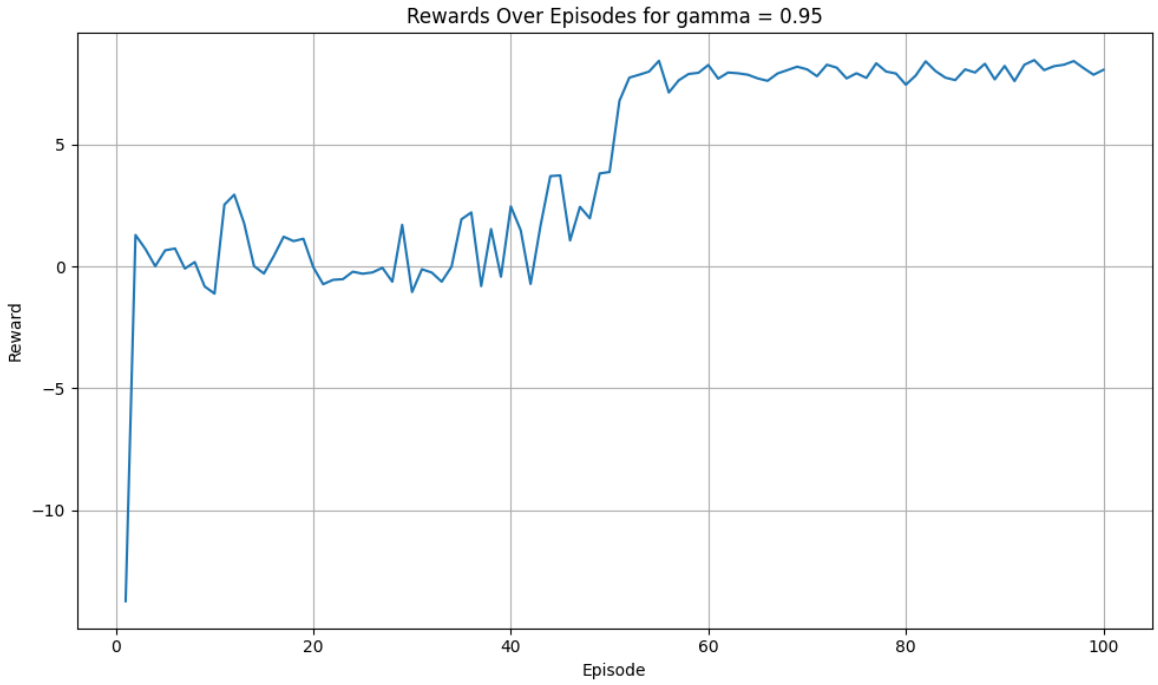

Figura 9: Recompensas para $\gamma = 0{,}95$

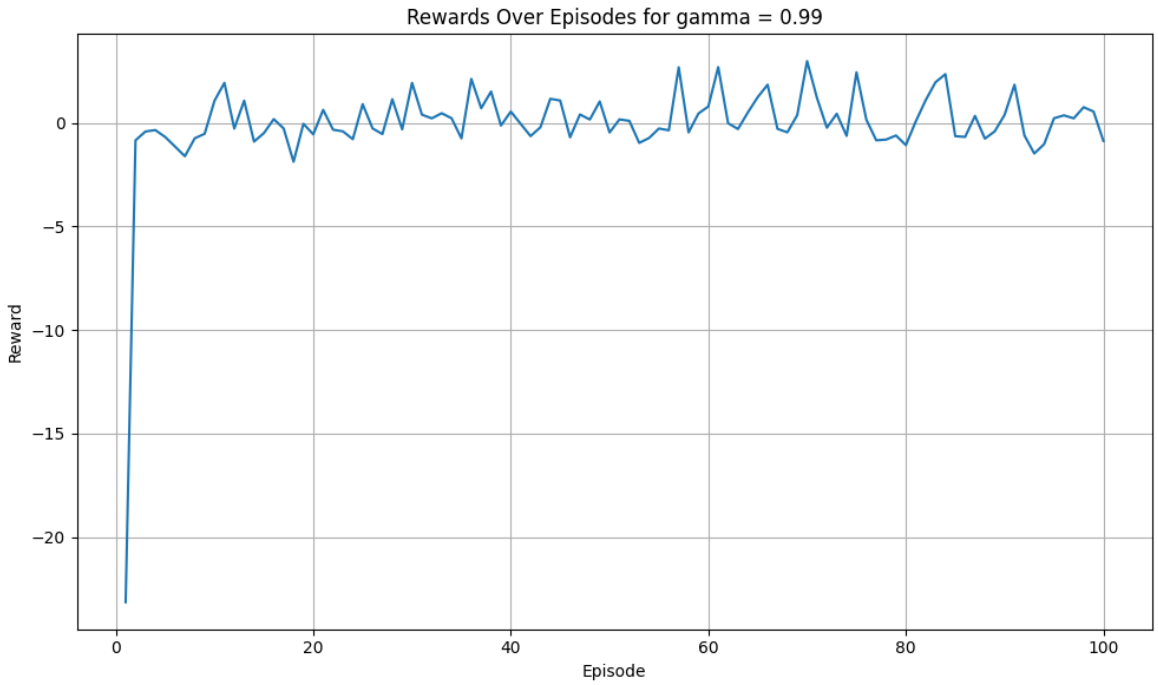

Figura 10: Recompensas para $\gamma = 0{,}99$

La primera observación que apreciamos es que todos los agentes comienzan obteniendo recompensas negativas. Esto se debe a que el agente está aprendiendo el entorno y, en este proceso, realiza una serie de movimientos aleatorios para modelar el entorno en el que interactúa. Al ser movimientos aleatorios, implica que siempre empieza con recompensas negativas. La segunda observación que tenemos es que los casos extremos no son tan buenos como los valores intermedios. Es decir, no se consiguen recompensas

tan altas como en los casos de valores intermedios.

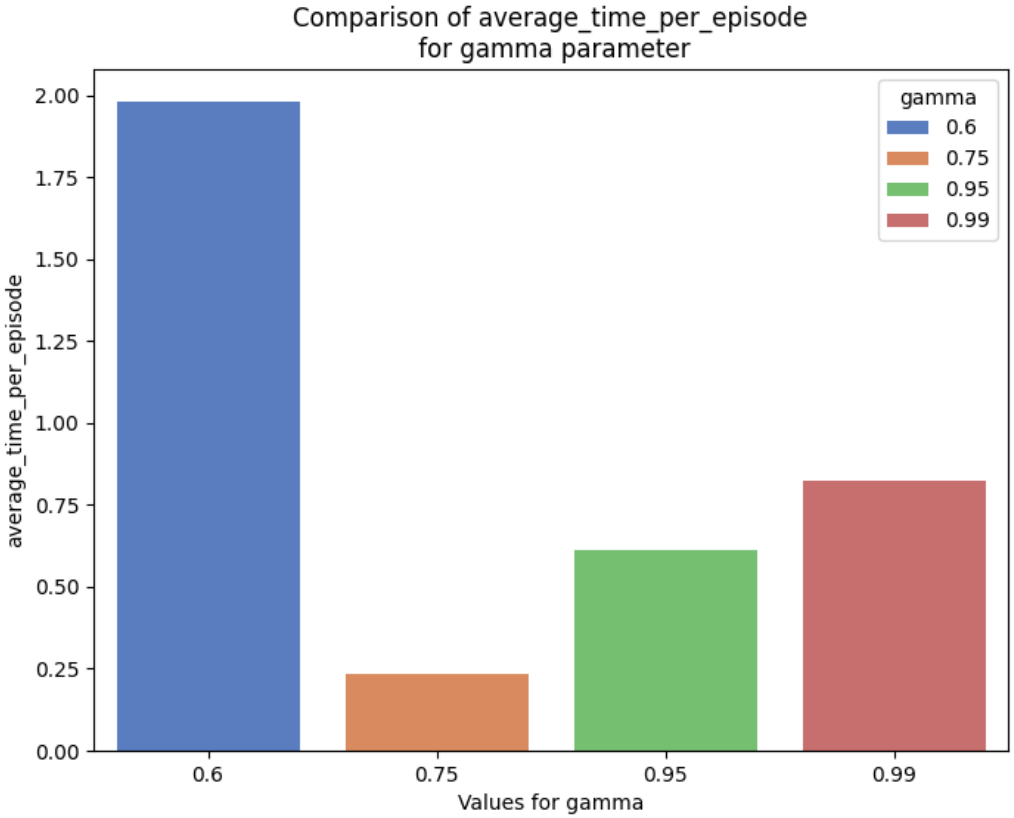

Figura 11: Promedio de tiempo por episodio por γ

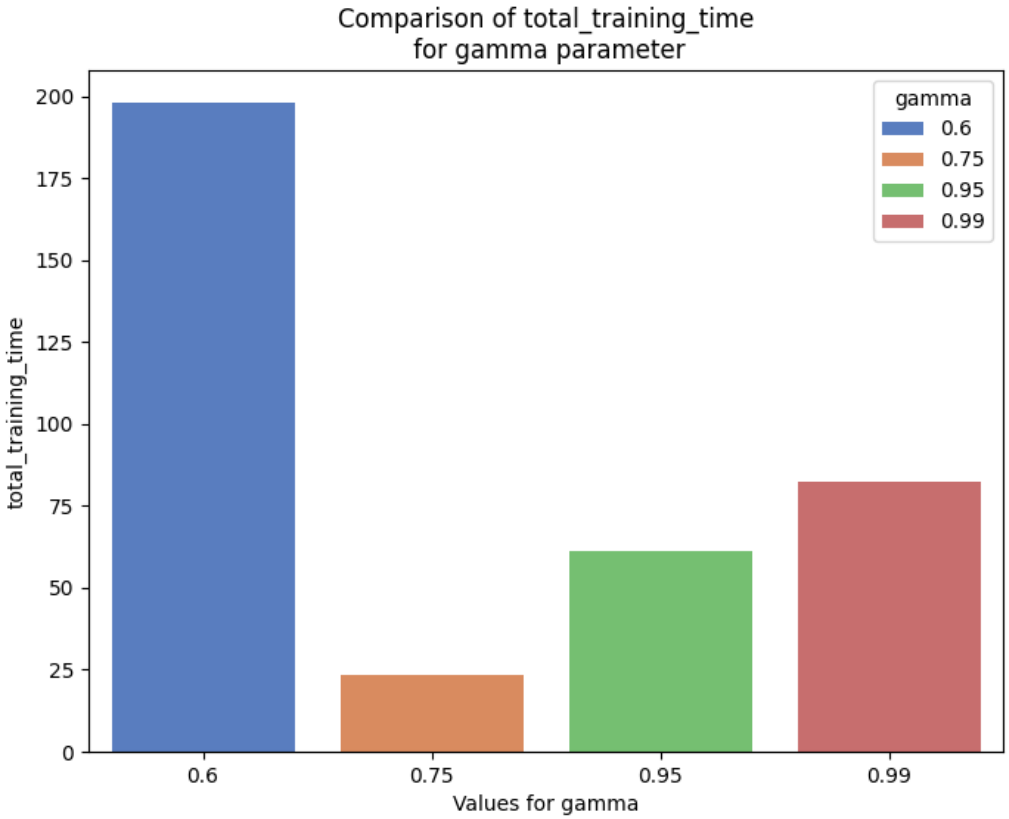

Figura 12: Promedio del tiempo total de la simulación (en segundos) por γ

Analizando la calidad de las ejecuciones, vemos que $\gamma = 0{,}6$ es un valor bastante ineficiente para el aprendizaje del agente, ya que el tiempo de ejecución por episodio es bastante elevado, lo que indica que el agente pierde mucho tiempo en otras alternativas.

Por otra parte, analicemos los casos de 0.75 y 0.99. Ambos permiten obtener una segunda mejor recompensa, pero vemos que si el agente considera demasiado las recompensas futuras, como es el caso de 0.99, puede que no obtenga la mayor recompensa a corto plazo y implique gastar más tiempo buscando o mejorando esta solución. Por lo tanto, descartamos 0.99 debido a su elevado coste computacional.

Quedan 0.75 y 0.95, ambos valores bastante buenos para este parámetro. Sin embargo, optaremos por un valor que nos dé una mayor recompensa en un tiempo de computación no muy elevado, como es el caso de 0.95. Aunque 0.75 obtiene una recompensa menor en un menor tiempo que 0.95, preferimos el equilibrio que ofrece 0.95.

En conclusión, nuestra hipótesis es correcta. No obstante, al hacer experimentos, nos hemos dado cuenta de que un valor demasiado próximo a 1 no favorece mucho en este entorno. Aun así, 0.95 es un número bastante próximo y permite obtener un mejor rendimiento en la simulación del agente en el entorno.

2. **Número de Trayectorias (num_trajectories)**: Este parámetro define el número de pasos aleatorios que el agente toma para explorar el entorno durante cada iteración de valor.

- **Hipótesis**: Un mayor número de trayectorias debería permitir al agente explorar más el entorno, recolectando más información y mejorando la precisión de las estimaciones de valor.

Las gráficas muestran la evolución de las recompensas sobre episodios con diferentes números de trayectorias: 100, 1000 y 10,000.

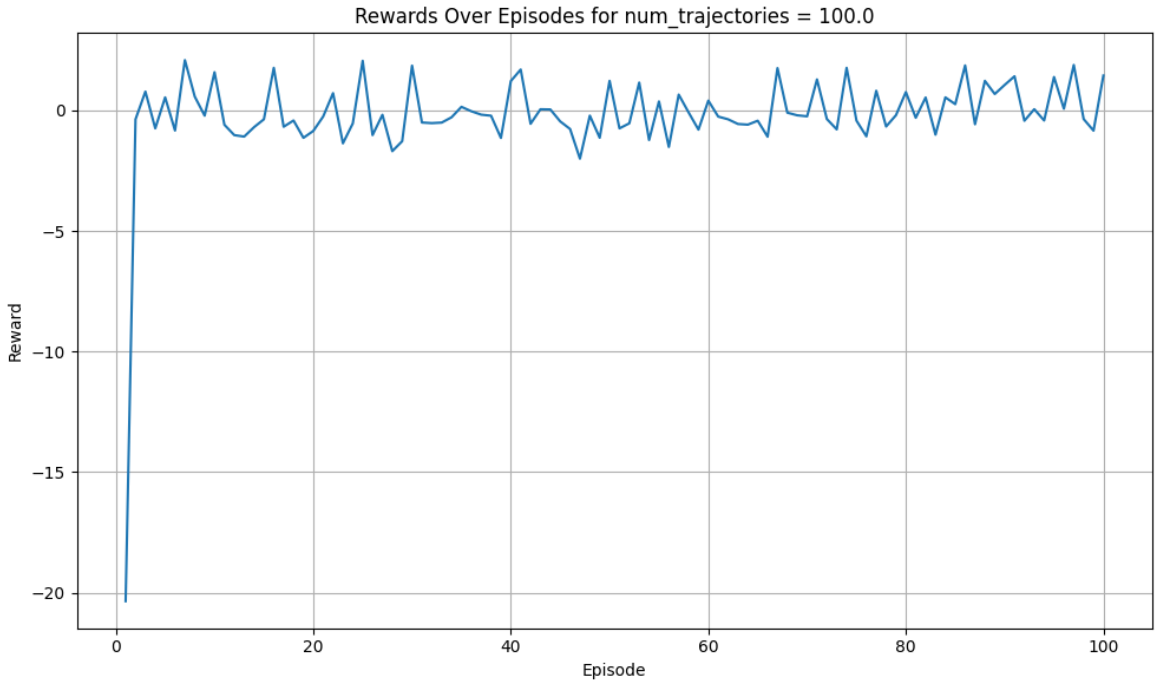

Figura 13: Recompensas para 100 trayectorias

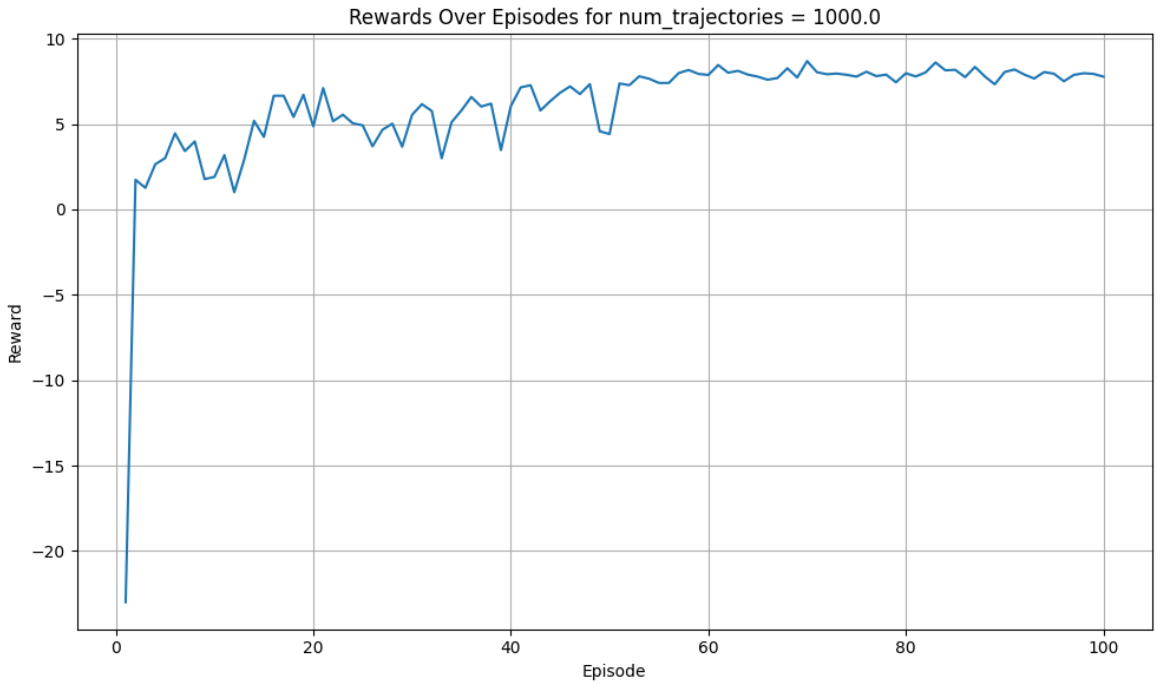

Figura 14: Recompensas para 1,000 trayectorias

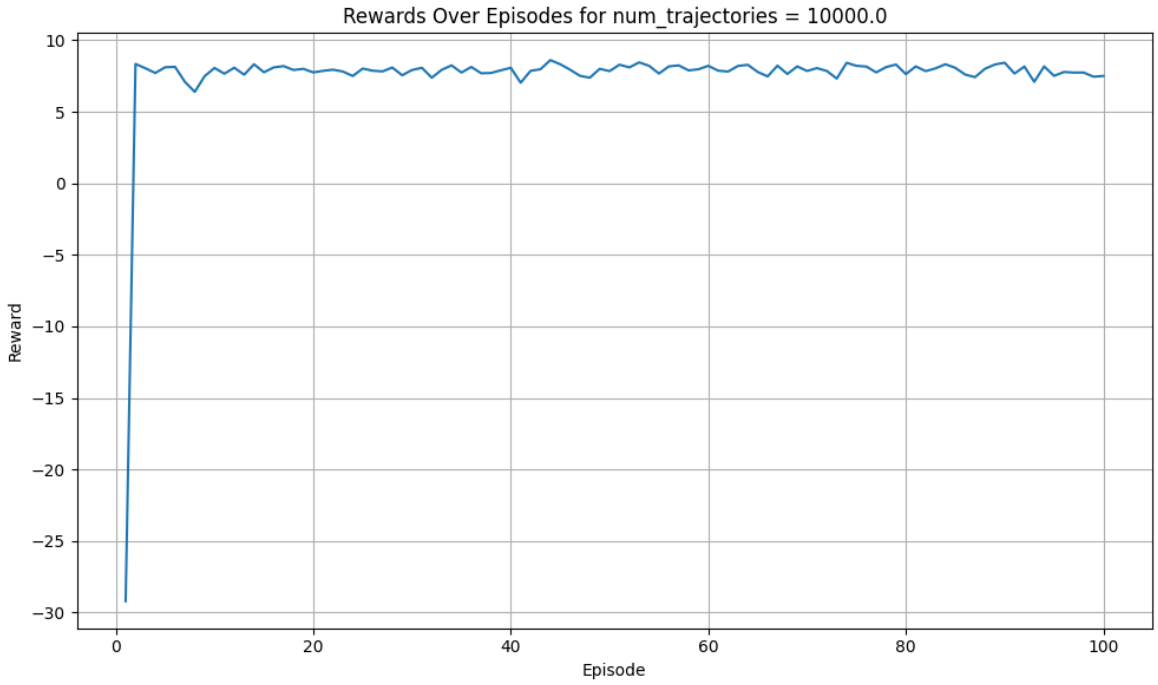

Figura 15: Recompensas para 10,000 trayectorias

1. **Estabilidad y Consistencia**: - Con 10,000 trayectorias (Figura 13), la línea es más recta y estable, mostrando recompensas consistentes y cercanas al valor óptimo. - Con 1,000 trayectorias (Figura 14), hay más variabilidad en las recompensas, aunque se alcanza una estabilidad razonable. - Con 100 trayectorias (Figura 20), la variabilidad es alta y las recompensas fluctúan significativamente.

Experimentalmente, se ha observado que al aumentar el número de trayectorias, el modelo dispone de más datos para aprender, lo que mejora la estabilidad y consistencia del rendimiento. Esto reduce la variabilidad y permite aproximarse a un modelo de mejor calidad.

Por otro lado, vamos a comparar las métricas de tiempo de ejecución. Dado que los tiempos promedio de ejecución por episodio son bastante similares, nos centraremos principalmente en el tiempo de entrenamiento.

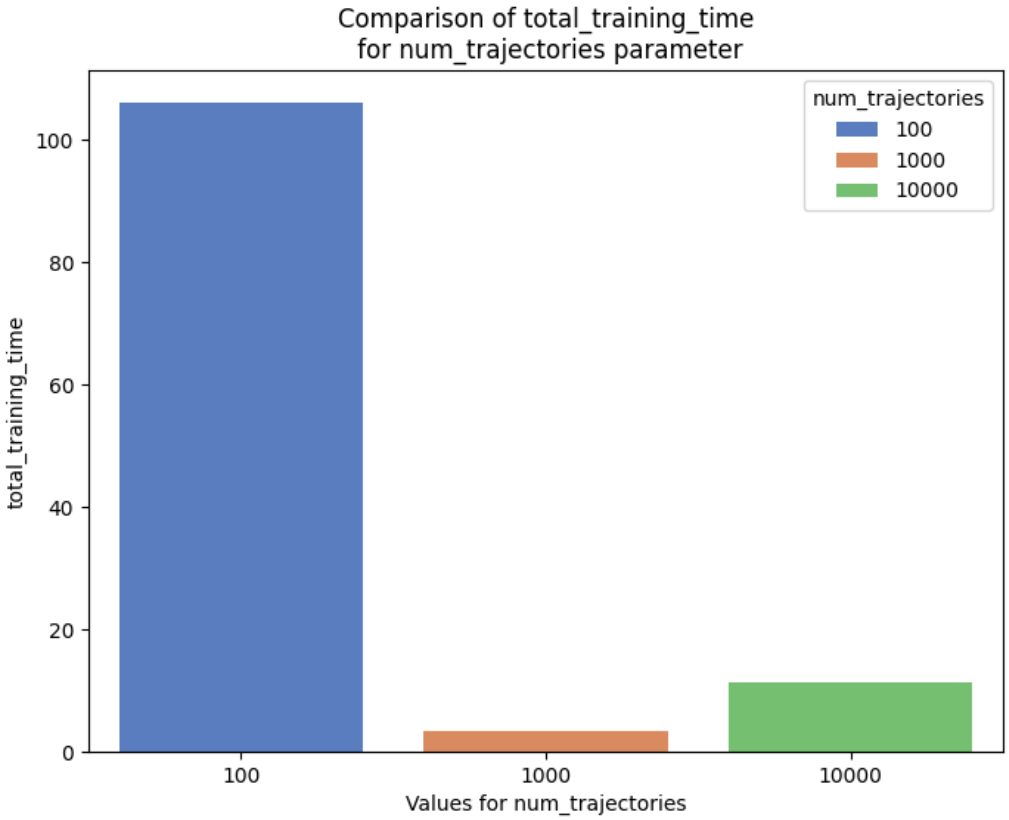

Figura 16: Tiempos de entrenamiento para cada `num_trajectories`

Vemos claramente que con pocos pasos aleatorios, el entrenamiento dura más. Esto puede deberse a que el modelo que se aproxima no es del todo preciso.

En conclusión, podemos afirmar que con un mayor número de pasos aleatorios, el algoritmo recopila más datos, lo que resulta en un modelo más realista al final del proceso.

3. **Umbral de Recompensa (reward_threshold)**: Este parámetro define el valor de recompensa deseado al cual queremos que el agente converja. En clase de teoría hemos visto la convergencia basada en la máxima diferencia entre valores de estados. Otra opción , la que se usa en la experimentaciones la de definir un umbral de recompensa a partir del cual parar.

- **Hipótesis**: A medida que el umbral aumenta, la recompensa que el agente puede llegar también debería aumentar.

Las gráficas muestran la evolución de las recompensas a lo largo de varios episodios con diferentes umbrales de recompensa: 0.05, 0.9, 5, y 8.5.

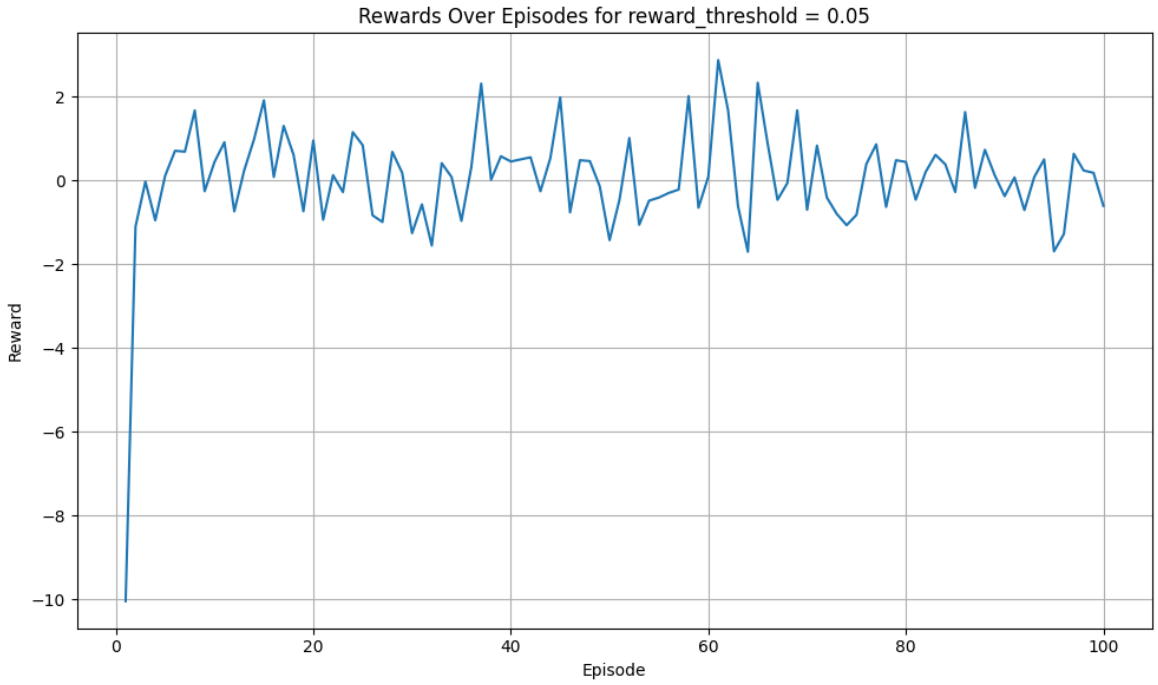

Figura 17: Recompensas con el umbral a 0.05

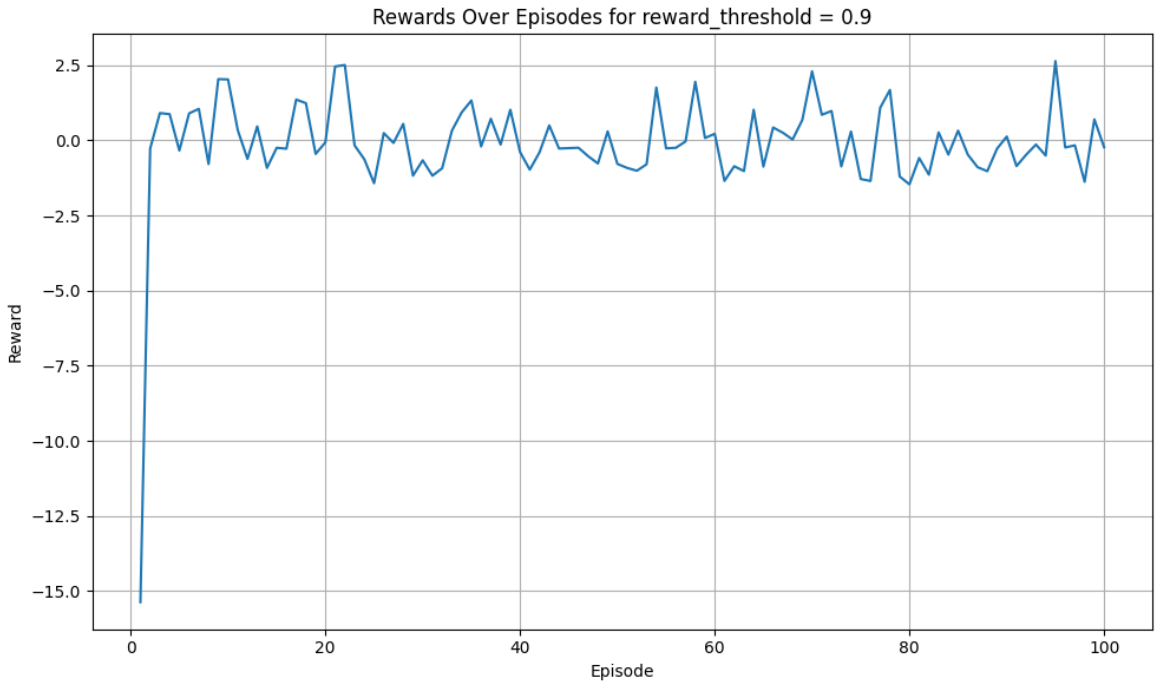

Figura 18: Recompensas con el umbral a 0.9

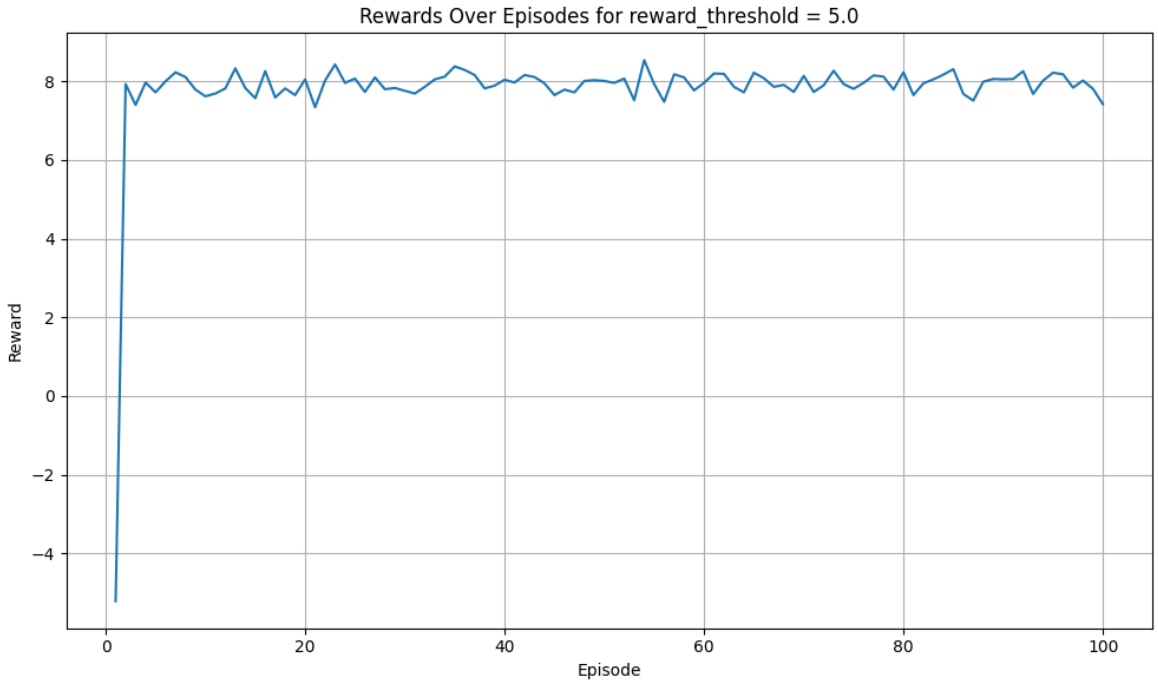

Figura 19: Recompensas con el umbral a 5

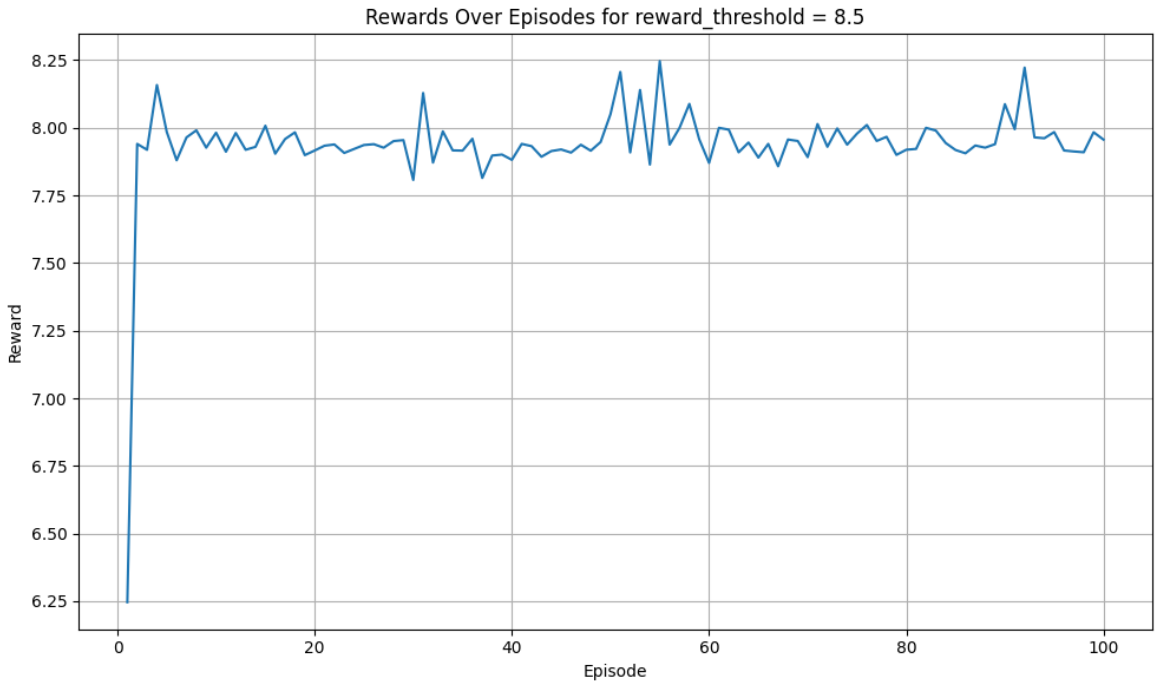

Figura 20: Recompensas con el umbral a 8.5

Observamos que, a medida que el umbral aumenta, las recompensas tienden a incrementarse. Sin embargo, a partir de un umbral de aproximadamente 8, las recompensas no muestran un aumento significativo.

Consideramos que esto se debe a que la recompensa máxima alcanzable está cerca de los 8 puntos. Por

lo tanto, la hipótesis de que las recompensas continúan aumentando indefinidamente con el umbral es incorrecta. Existe un límite de umbral alrededor del valor de 8.

Procederemos a realizar experimentos variando estos parámetros uno a uno, manteniendo los demás constantes, para evaluar su impacto en el rendimiento del agente y confirmar o refutar nuestras hipótesis.

4. Q-Learning

Este algoritmo que ayuda a un agente a aprender la mejor forma de actuar en un entorno para maximizar la recompensa acumulada a lo largo del tiempo. El agente usa una tabla llamada Q-Table para estimar el valor de tomar una acción en un estado determinado.

4.1. Implementación

Q-Table: Es una tabla que almacena los valores Q, donde cada valor Q representa la recompensa esperada de tomar una acción en un estado específico y seguir la política óptima.

Las funciones principales del agente que implementa el algoritmo Q-Learning son las siguientes:

select_action: Selecciona la acción en un estado dado. Explora con probabilidad `epsilon` o elige la mejor acción según la Q-Table.

update_Q: Actualiza la Q-Table basándose en la acción tomada y la recompensa recibida. Calcula la diferencia temporal (TD error) y ajusta el valor Q.

learn_from_episode: Realiza un episodio completo de entrenamiento. Resetea el entorno, selecciona acciones, actualiza la Q-Table y aplica el decaimiento de parámetros.

test_episode: Ejecuta un episodio de prueba para evaluar el rendimiento del agente. Selecciona acciones basadas en la política actual (sin exploración).

policy: Genera la política óptima basada en la Q-Table. Selecciona la mejor acción para cada estado.

decay_parameters: Reduce gradualmente `epsilon` y `learning_rate`, manteniendo los valores por encima de los mínimos especificados.

Primero, el agente se inicializa con el entorno y los parámetros (`__init__`), creando una Q-Table vacía. Durante el entrenamiento, el agente realiza múltiples episodios usando `learn_from_episode`. En cada episodio, se resetea el entorno y se seleccionan acciones basadas en `select_action`. Las acciones se ejecutan, se reciben recompensas, y se actualiza la Q-Table con `update_Q`. Al final de cada episodio, se aplica el decaimiento a `epsilon` y `learning_rate` con `decay_parameters`.

Para evaluar el rendimiento, se ejecuta `test_episode`, donde se mide la recompensa total obtenida sin exploración. Finalmente, la función `policy` genera la política óptima basada en la Q-Table, seleccionando la mejor acción para cada estado.

4.2. Experimentación y resultados

De forma similar a los experimentos anteriores, para evaluar cómo influyen los hiperparámetros sobre la calidad del agente, procederemos de la manera más sencilla:

- Fijaremos valores constantes para todos los hiperparámetros excepto con el que vayamos a experimentar.
- Estas constantes iniciales son valores que más o menos funcionan bien para el algoritmo y para el hiperparámetro con el que experimentaremos, probaremos rangos de valores que consideremos adecuados y confirmaremos o refutaremos nuestras hipótesis.

Antes de empezar con la experimentación, tenemos que saber que hiperparámetros posee Q-Learning, que son los siguientes:

- gamma
- learning_rate
- epsilon
- learning_rate_decay
- epsilon_decay

Los hiperparámetros con los que experimentaremos serán: gamma, epsilon, learning_rate, epsilon_decay y learning_rate_decay.

1. **Factor de Descuento (`gamma`)**: Determina la importancia de las recompensas futuras. Un valor cercano a 1 hace que las recompensas futuras sean importantes.

   - **Hipótesis**: Un valor alto de `gamma` (cercano a 1) permitirá al agente considerar más las recompensas futuras, lo cual debería mejorar el rendimiento general del agente al hacer que planifique a largo plazo.

A continuación, se presentan las gráficas de las recompensas del agente con diferentes valores de `gamma`:

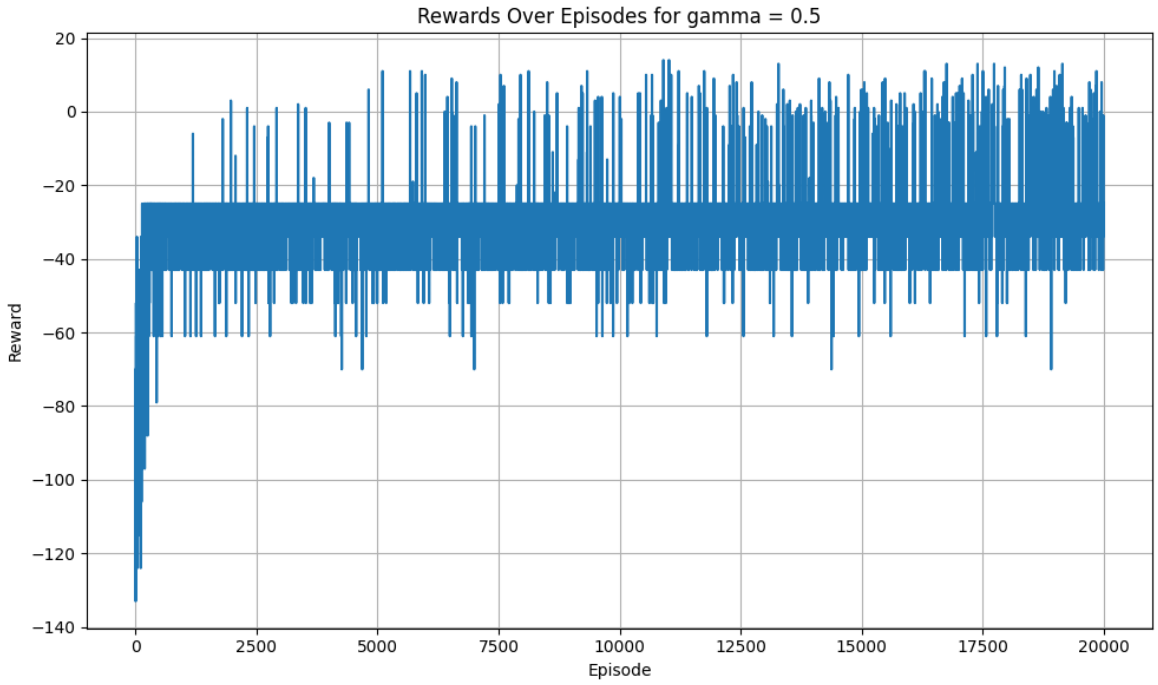

Figura 21: Recompensas con gamma 0.5

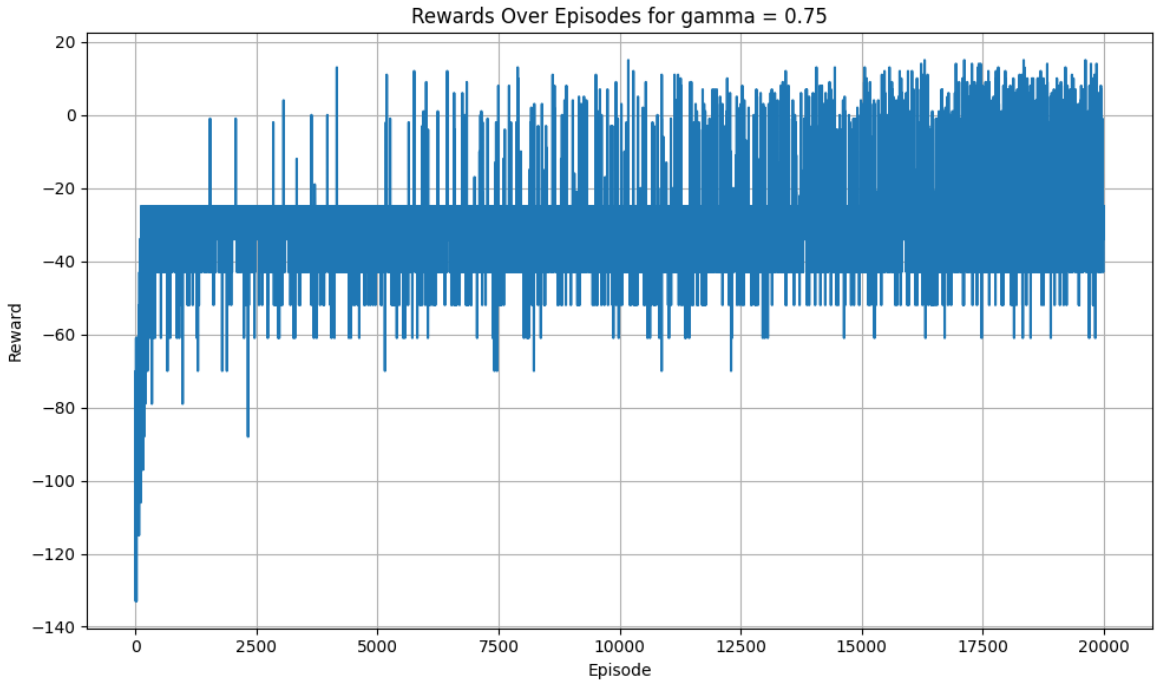

Figura 22: Recompensas con gamma 0.75

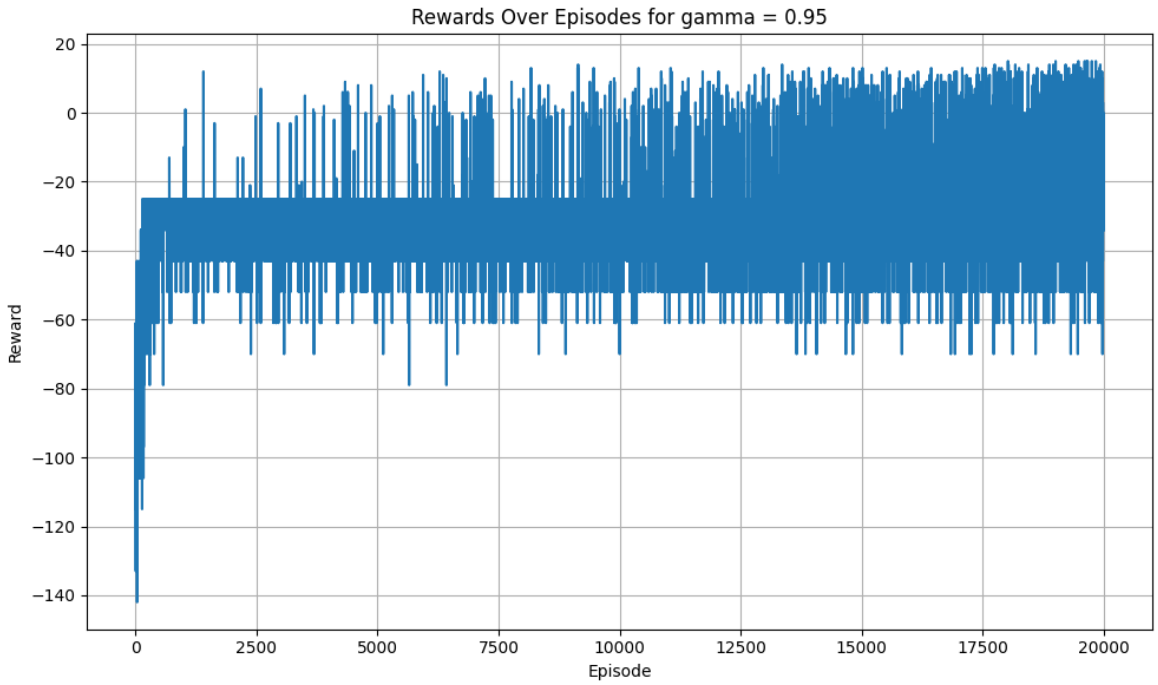

Figura 23: Recompensas con gamma 0.95

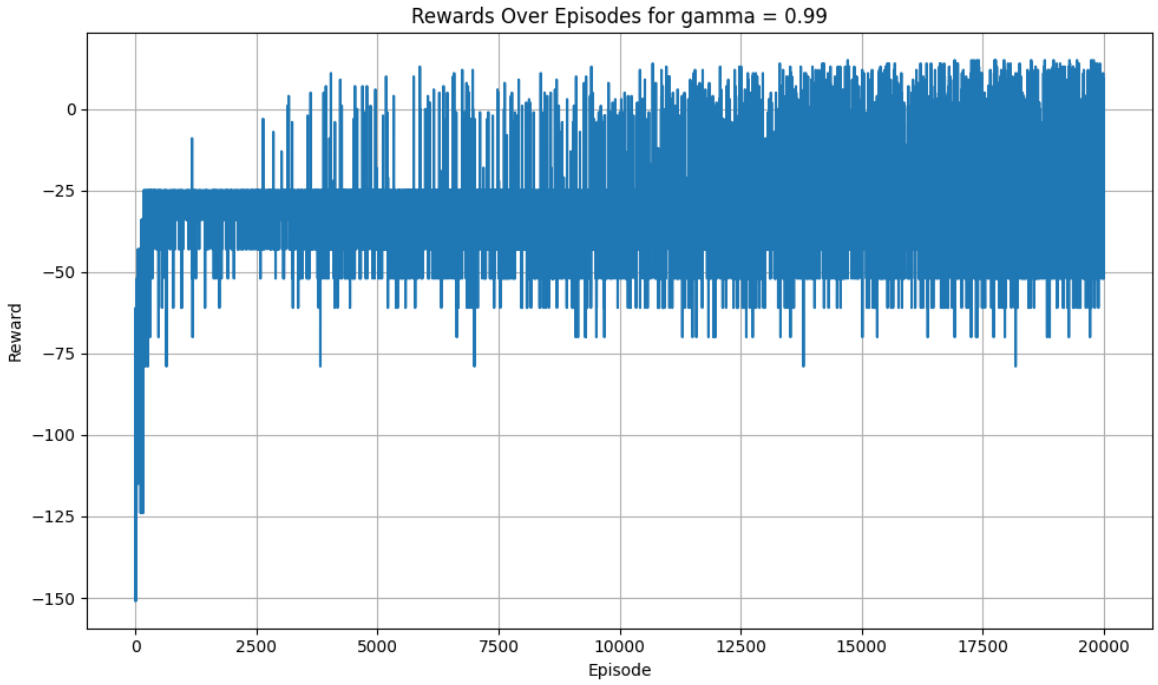

Figura 24: Recompensas con gamma 0.99

En cuanto a las métricas de tiempos de entrenamientos y las recompensas dependiendo del valor de `gamma`, se presentan las siguientes gráficas:

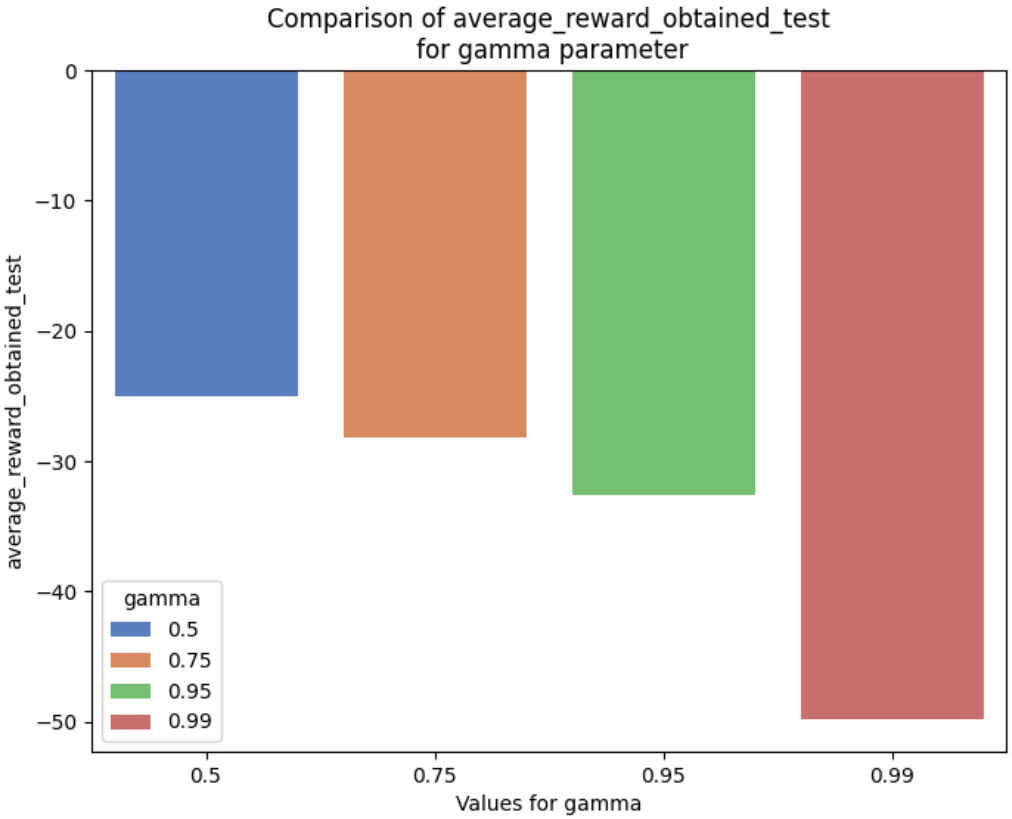

Figura 25: Recompensas promedio obtenidas en las pruebas en comparación para diferentes valores de gamma

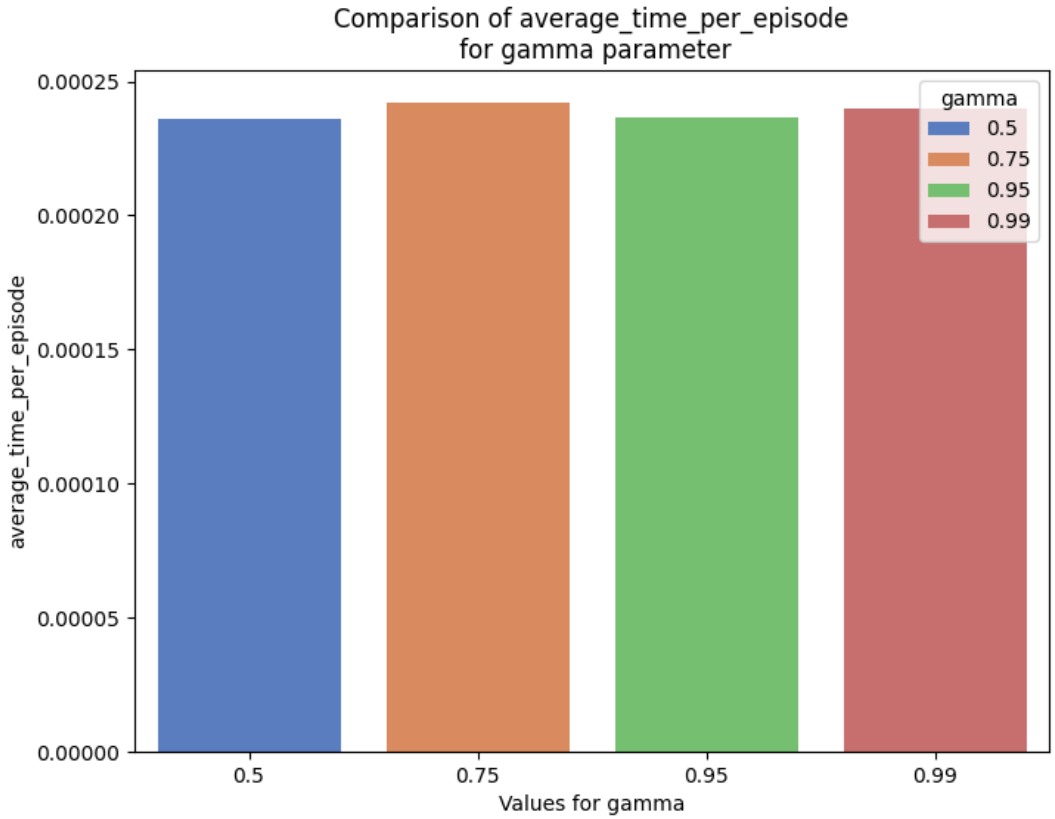

Figura 26: Tiempo promedio por episodio de entrenamiento en comparación para diferentes valores de gamma

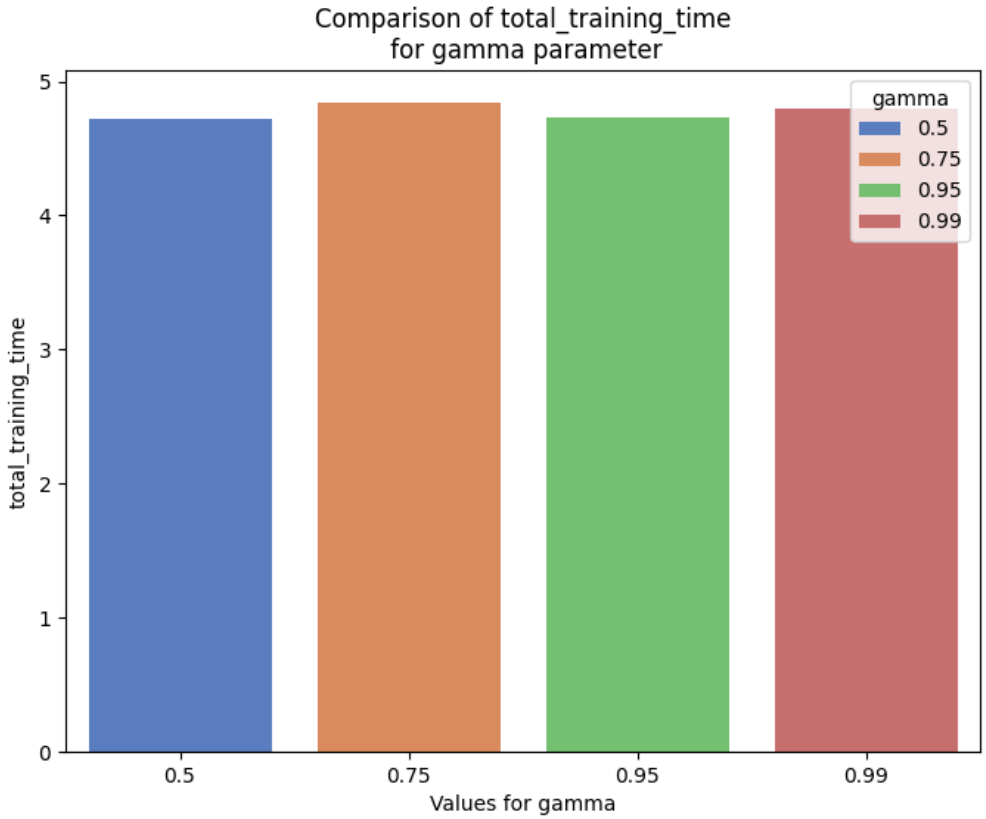

Figura 27: Tiempo total de entrenamiento en comparación para diferentes valores de gamma

Vemos que para valores más altos de `gamma` (0.95 y 0.99), el agente tiende a obtener recompensas más consistentes y menos negativas a lo largo del tiempo. En particular, `gamma` = 0.99 muestra una menor variabilidad y una tendencia hacia recompensas cercanas a 0. En contraste, los valores más bajos de `gamma` (0.5 y 0.75) muestran una mayor variabilidad y recompensas más negativas en promedio. Además, el tiempo promedio por episodio y el tiempo total de entrenamiento aumentan con valores más altos de `gamma`.

Por lo tanto, concluimos que un valor más alto de `gamma` mejora la estabilidad y el rendimiento a largo plazo del agente en términos de recompensas obtenidas, aunque esto viene a costa de un mayor tiempo de entrenamiento.

Respecto a los tiempos de entrenamiento, observamos que, independientemente de la gamma, los valores son similares. Sin embargo, en la figura 26 se puede ver que a medida que aumenta la gamma, mayor es la media de recompensas, lo cual confirma la hipótesis.

2. **Tasa de Aprendizaje (`learning_rate`)**: Controla cuánto se ajusta el valor Q basado en la nueva información. Un valor alto puede hacer que el aprendizaje sea rápido pero inestable.

- **Hipótesis**: Una tasa de aprendizaje adecuada permitirá al agente aprender de manera efectiva sin sobreajustar a los datos, lo que podría mejorar su rendimiento en la tarea.

A continuación, se presentan las gráficas de las recompensas del agente con diferentes valores de `learning_rate`:

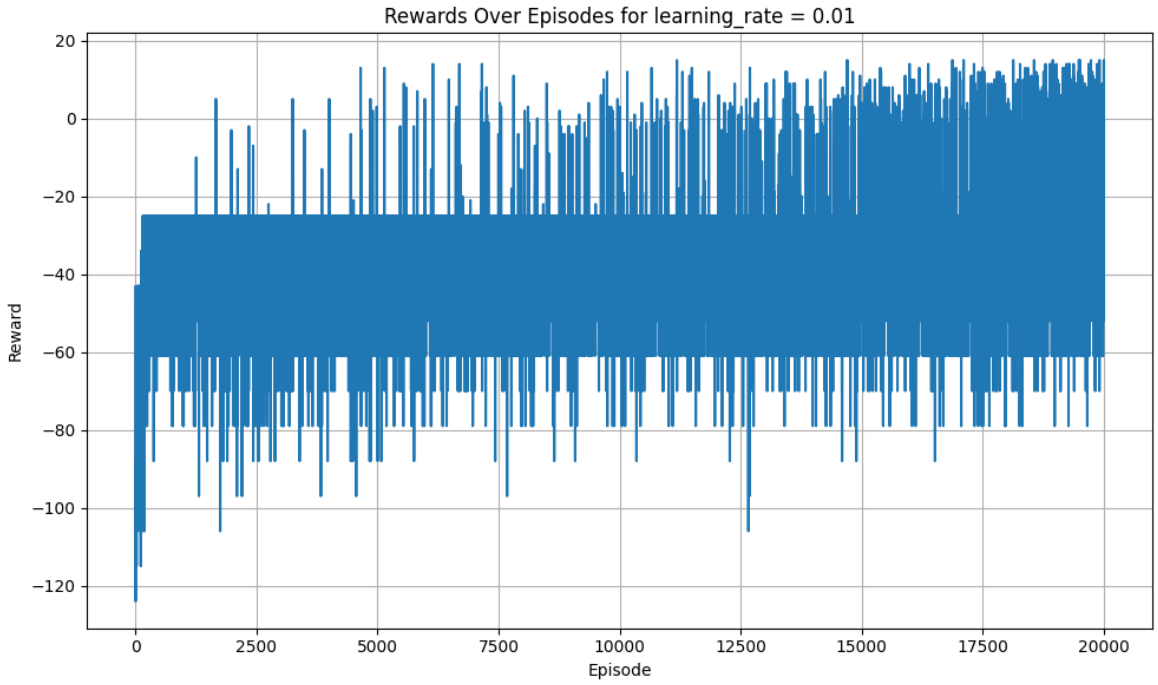

Figura 28: Recompensas con learning rate 0.01

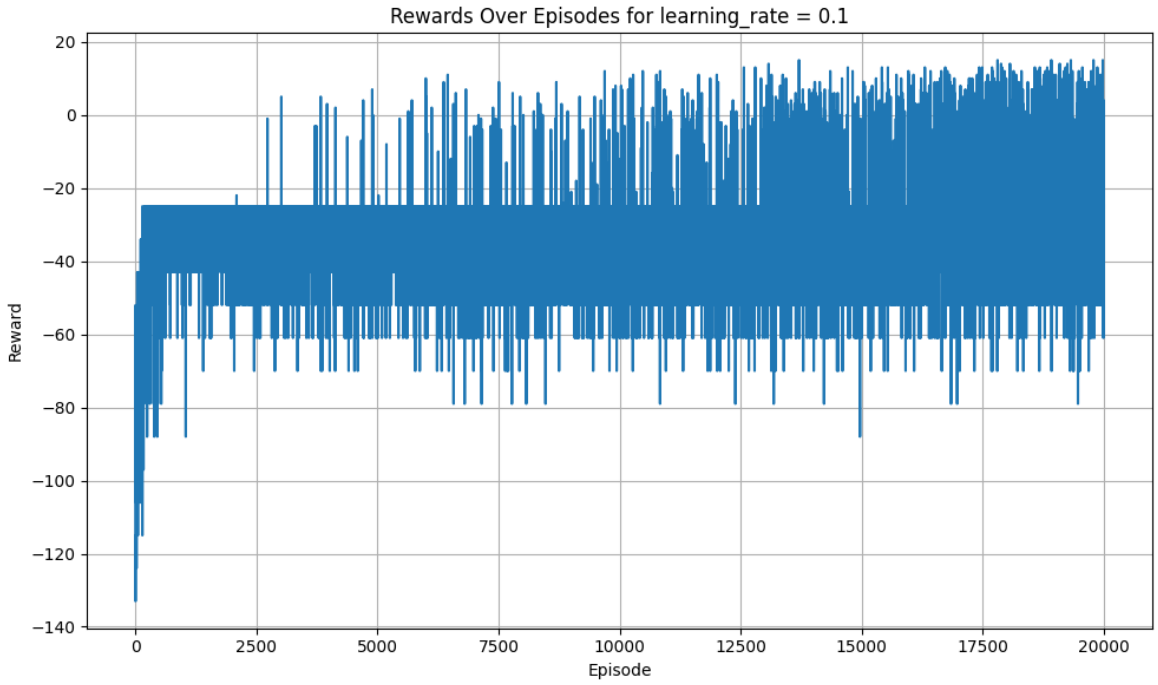

Figura 29: Recompensas con learning rate 0.1

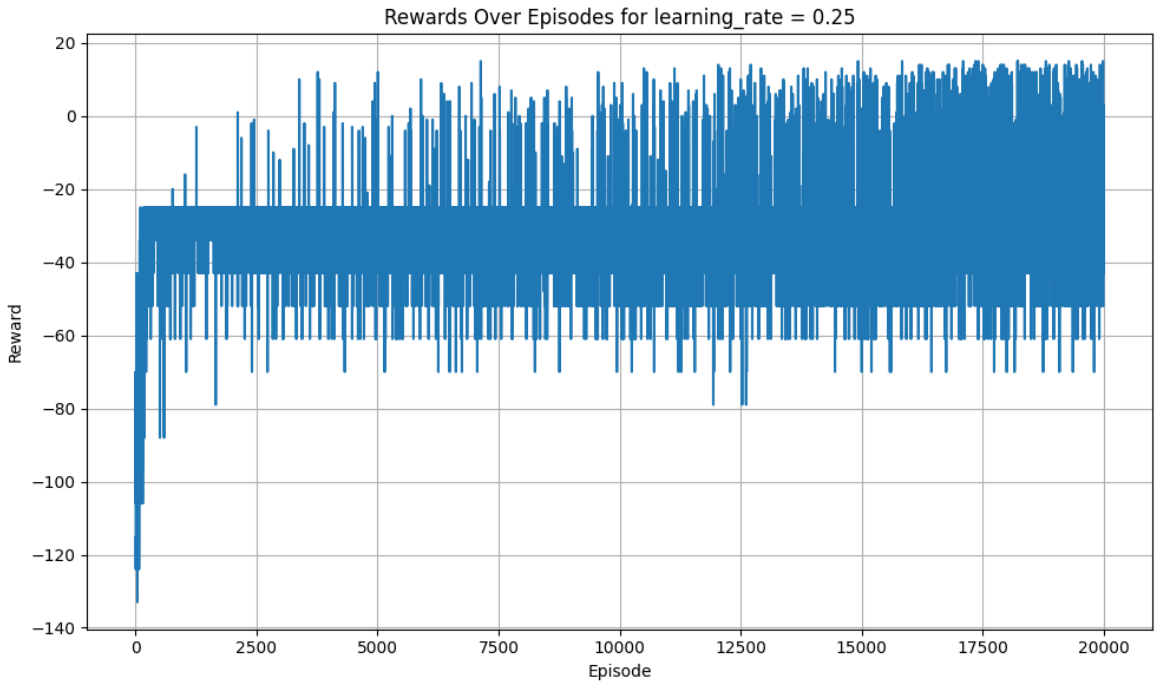

Figura 30: Recompensas con learning rate 0.25

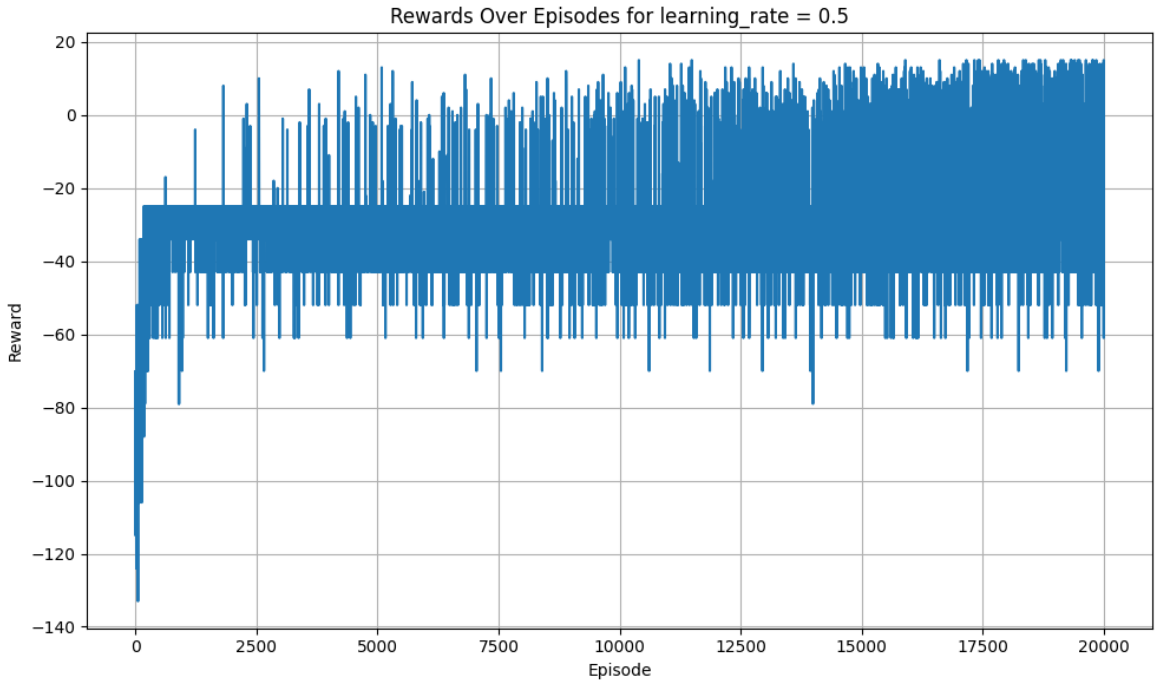

Figura 31: Recompensas con learning rate 0.5

En cuanto a las métricas de tiempos de entrenamiento y las recompensas dependiendo del valor de `learning_rate`, se presentan las siguientes gráficas:

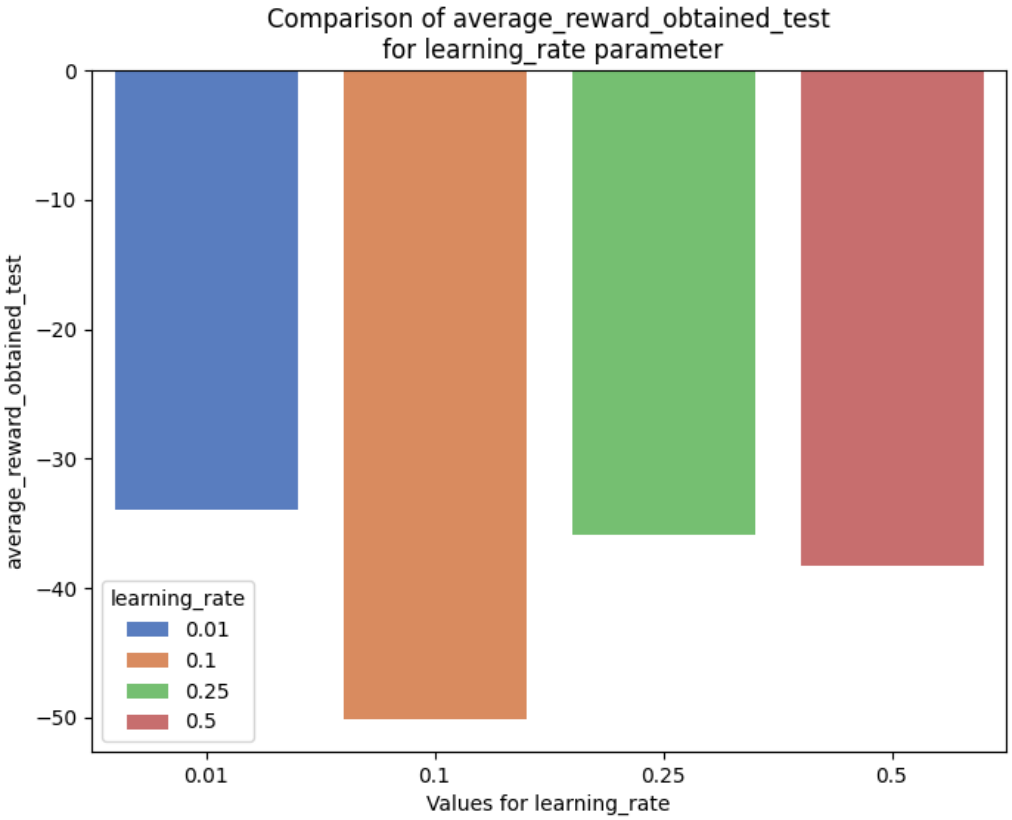

Figura 32: Recompensas promedio obtenidas en las pruebas en comparación para diferentes valores de learning rate

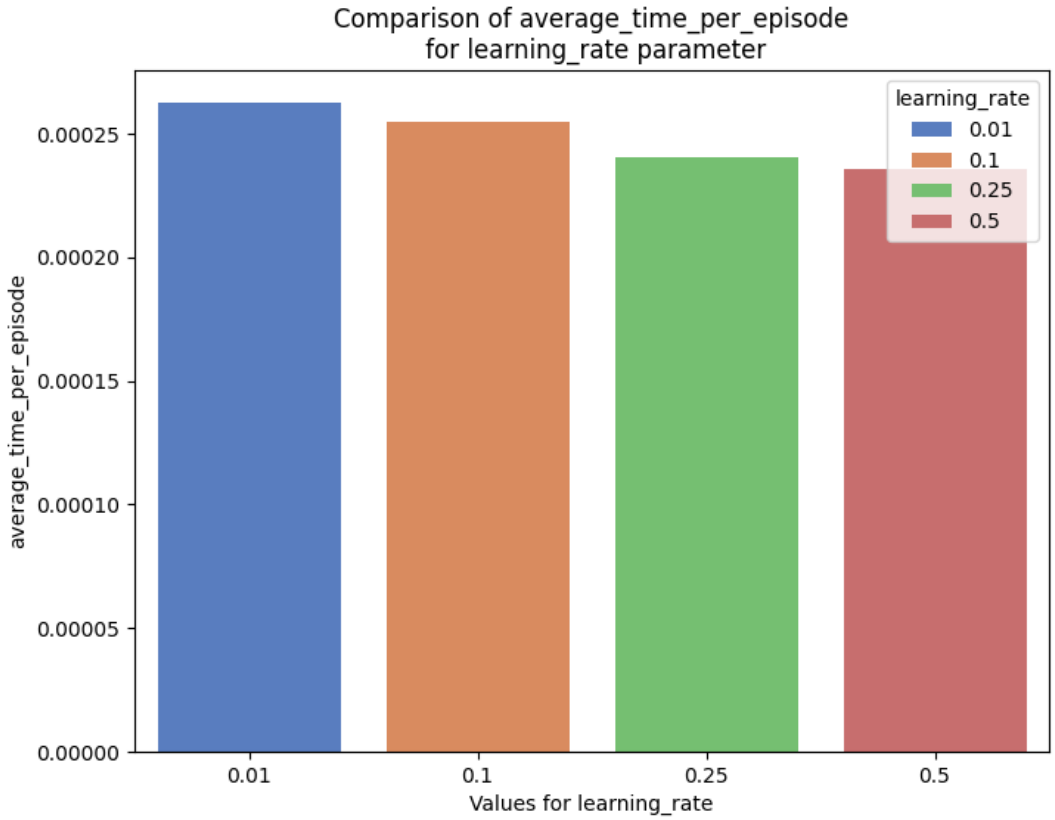

Figura 33: Tiempo promedio por episodio de entrenamiento en comparación para diferentes valores de learning rate

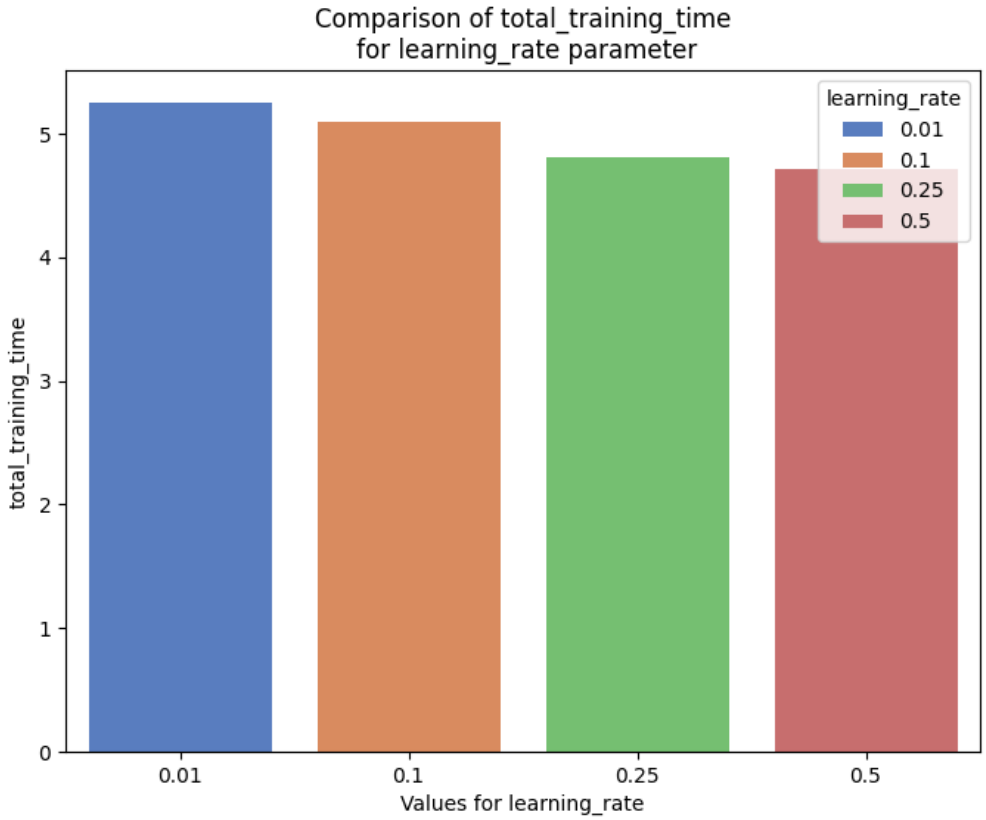

Figura 34: Tiempo total de entrenamiento en comparación para diferentes valores de learning rate

Vemos que, igualmente como en el caso anterior, los tiempos de entrenamiento y la duración de los episodios no difieren significativamente. Sin embargo, observamos que con una tasa de aprendizaje de 0.1 se obtienen las mejores recompensas.

Por lo tanto, concluimos que no hay una relación lineal entre la recompensa y la tasa de aprendizaje. En este caso, la mejor tasa de aprendizaje está alrededor de 0.1.

3. **Exploración-Explotación** (`epsilon`): Parámetro de exploración-explotación, define la probabilidad de que el agente seleccione una acción aleatoria para explorar el entorno. Si aumenta este valor, el agente hará menos explotación y más exploración. Consideramos que cuanto más explore, mejores recompensas obtendrá.

- **Hipótesis**: A medida que reducimos el epsilon, habrá más explotación, lo que limita la exploración y no deja margen para llegar a nuevos estados. `epsilon`

A continuación, se presentan las gráficas de las recompensas del agente con diferentes valores de `epsilon`:

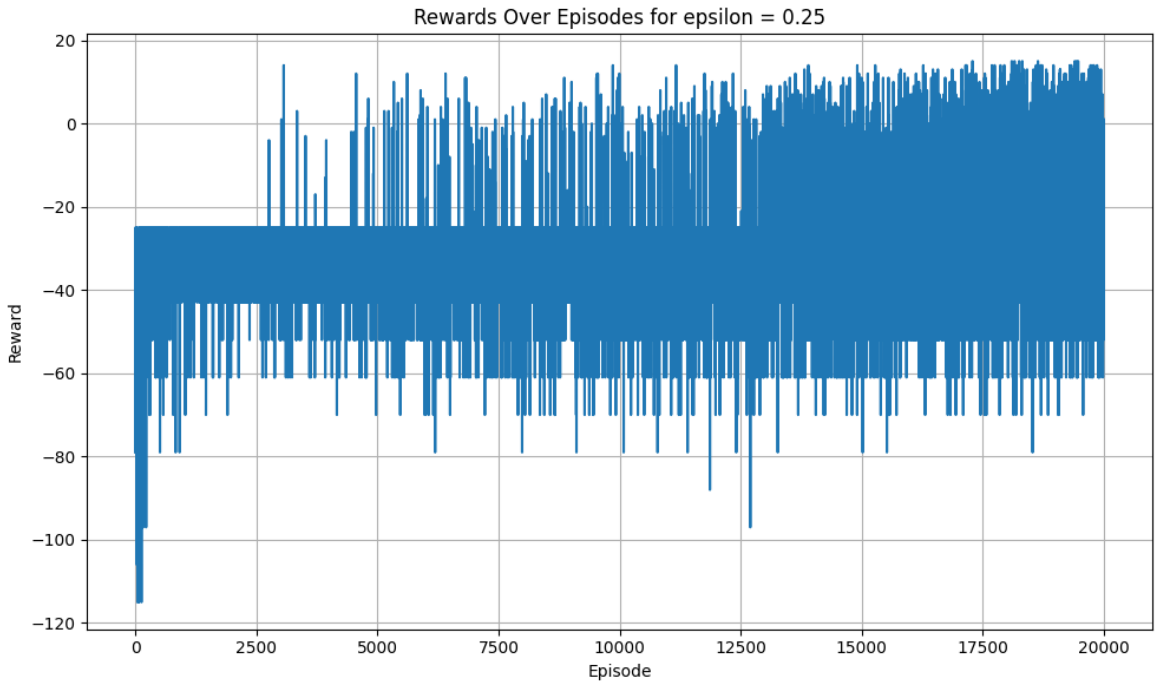

Figura 35: Recompensas con epsilon 0.25

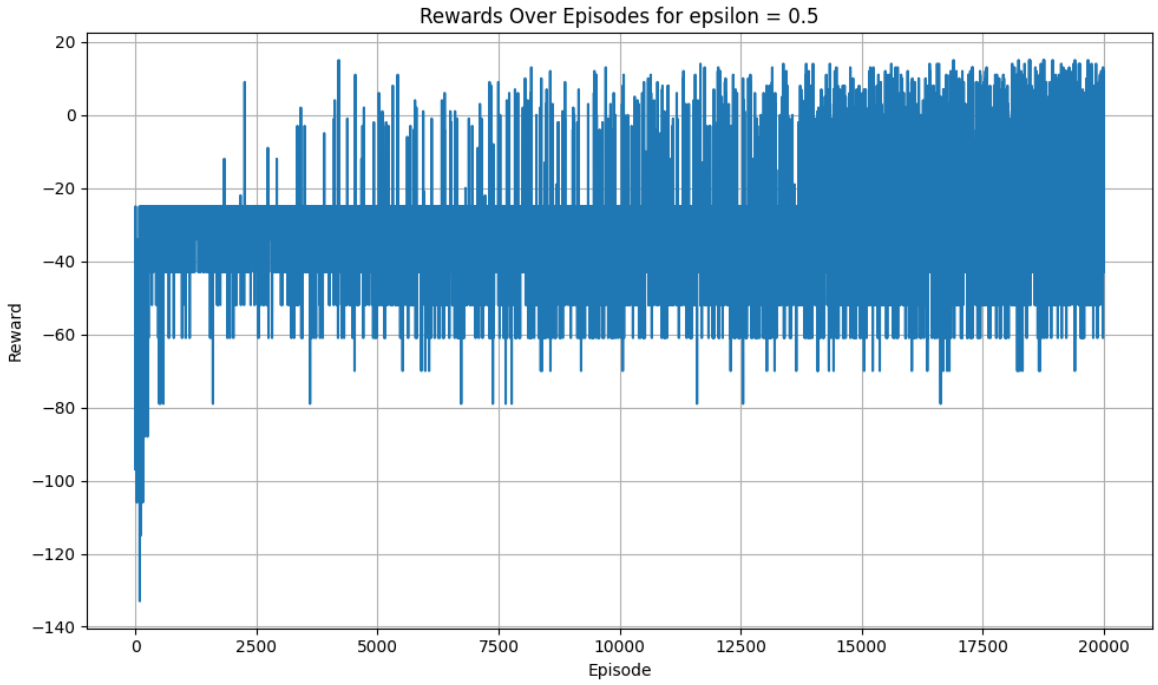

Figura 36: Recompensas con epsilon 0.5

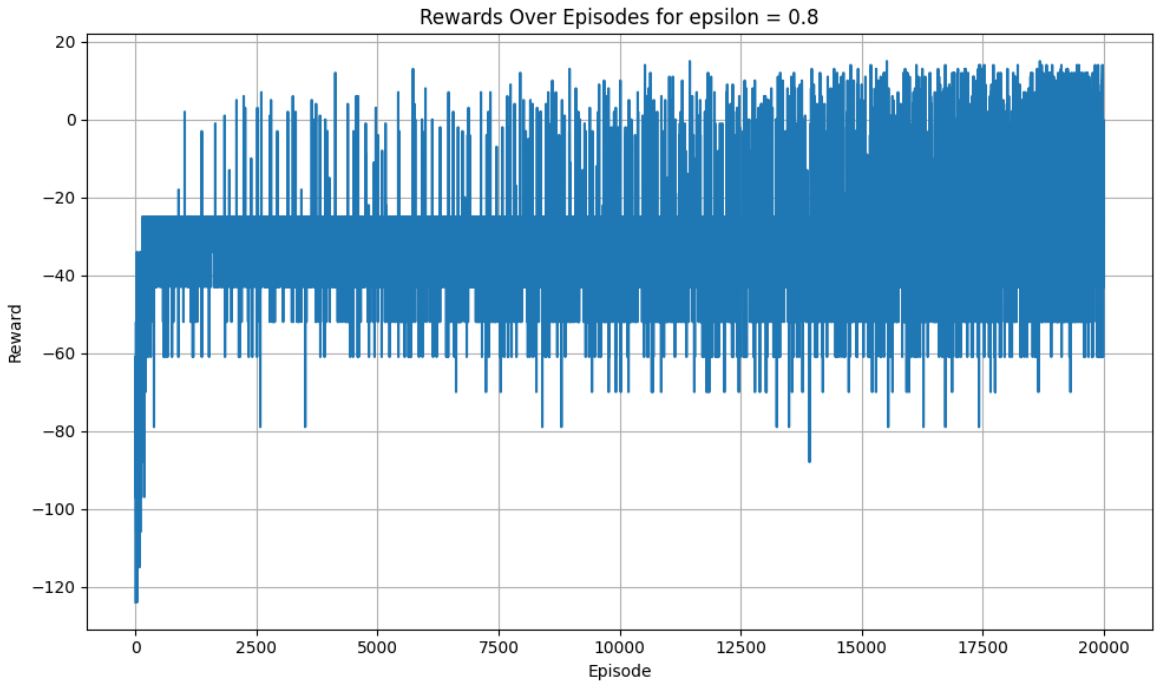

Figura 37: Recompensas con epsilon 0.8

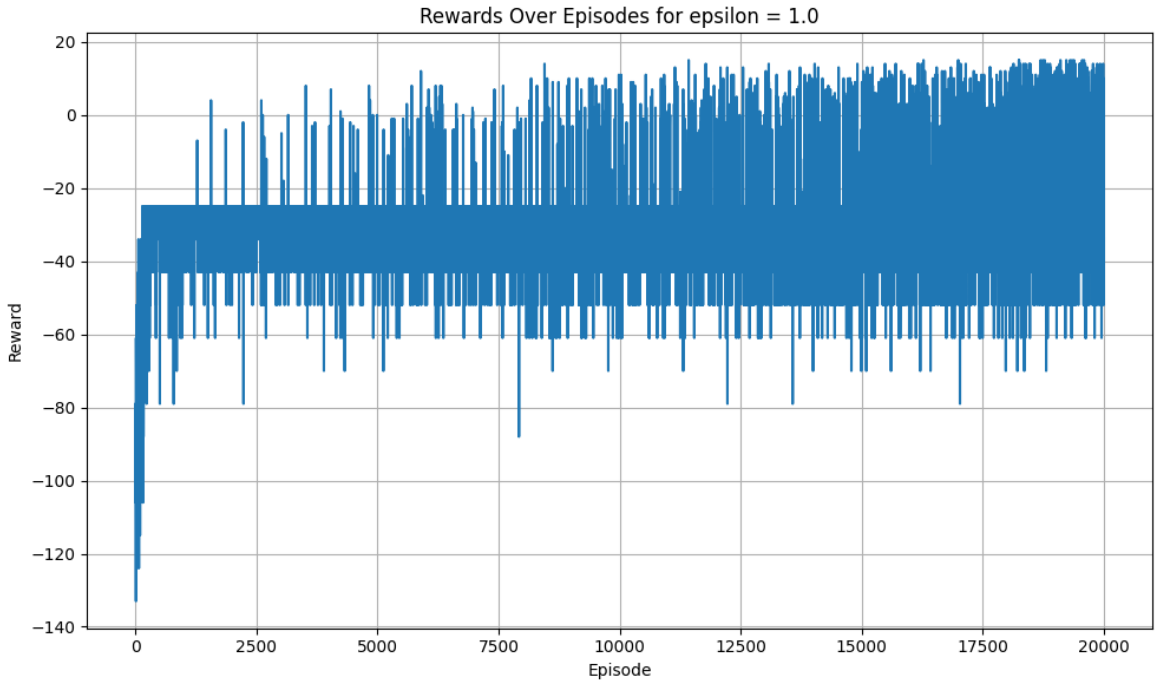

Figura 38: Recompensas con epsilon 1.0

En cuanto a las métricas de tiempos de entrenamiento y las recompensas dependiendo del valor de `epsilon`, se presentan las siguientes gráficas:

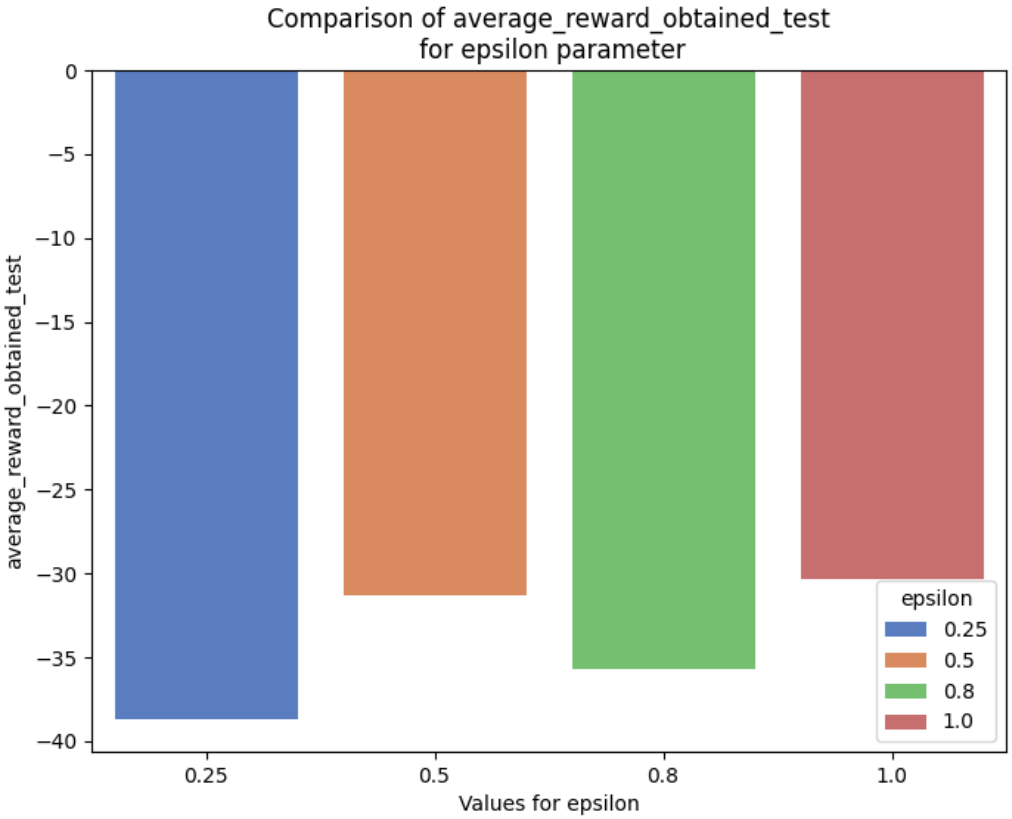

Figura 39: Recompensas promedio obtenidas en las pruebas en comparación para diferentes valores de epsilon

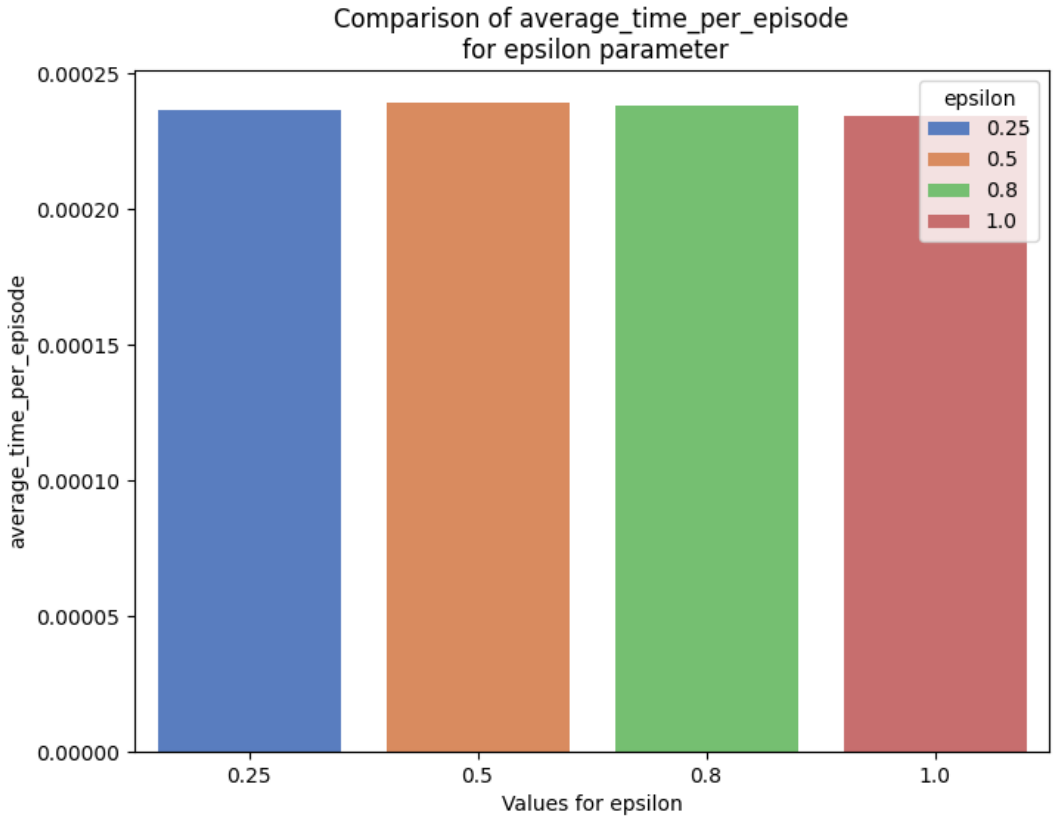

Figura 40: Tiempo promedio por episodio de entrenamiento en comparación para diferentes valores de epsilon

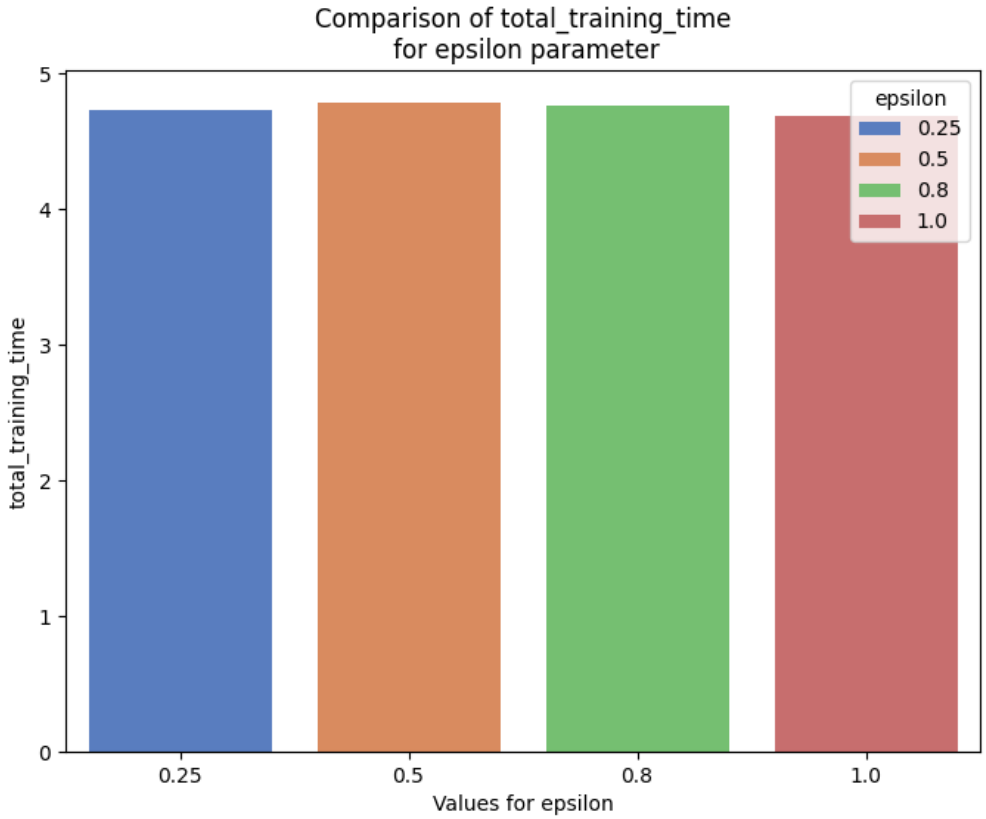

Figura 41: Tiempo total de entrenamiento en comparación para diferentes valores de epsilon

Vemos en las tres últimas gráficas que el tiempo de ejecución de episodios no influye significativamente, y las recompensas tampoco parecen seguir un patrón claro.

Por lo tanto, concluimos, basados en los experimentos, que el valor de epsilon con un valor intermedio (0.5) puede ser válido para este parámetro.

4. **Decaimiento de la Tasa de Aprendizaje** (`learning_rate_decay`): Factores que reducen gradualmente `learning_rate` a medida que el agente acumula más experiencia.

- **Hipótesis**: Un decaimiento gradual de la tasa de aprendizaje puede ayudar al agente a explorar suficientemente el entorno al principio y luego explotar más su conocimiento a medida que acumula más experiencia, lo que podría mejorar su rendimiento en términos de recompensas acumuladas.

A continuación, se presentan las gráficas de las recompensas del agente con diferentes valores de `learning_rate_decay`:

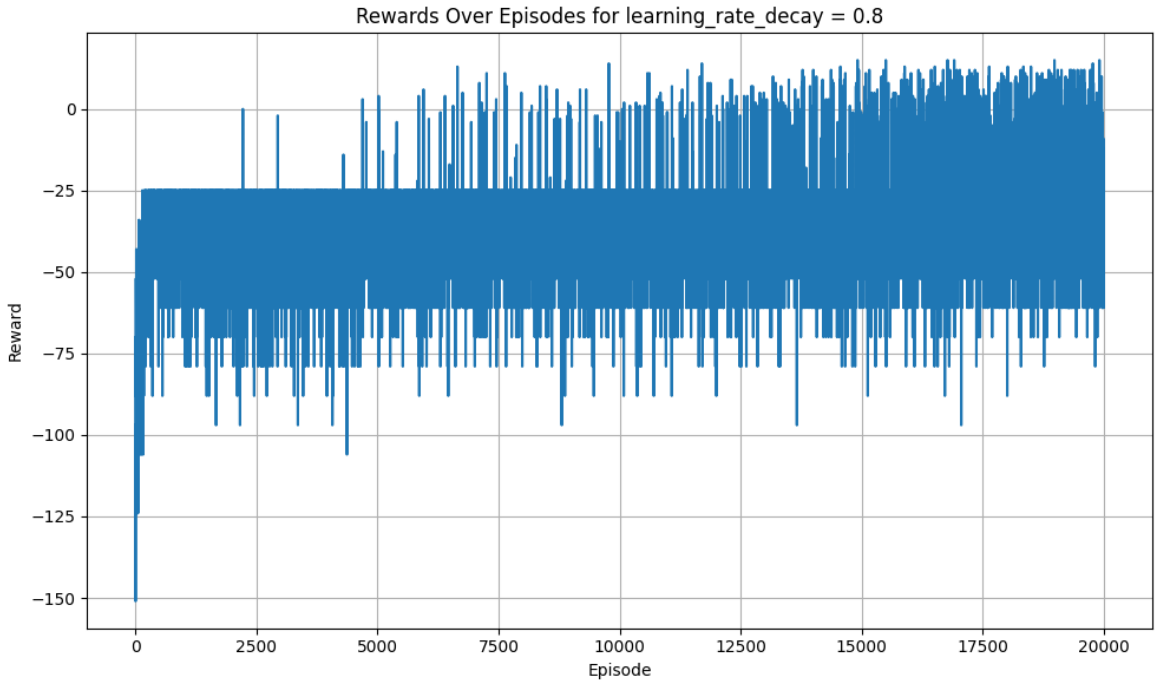

Figura 42: Recompensas con learning rate decay 0.8

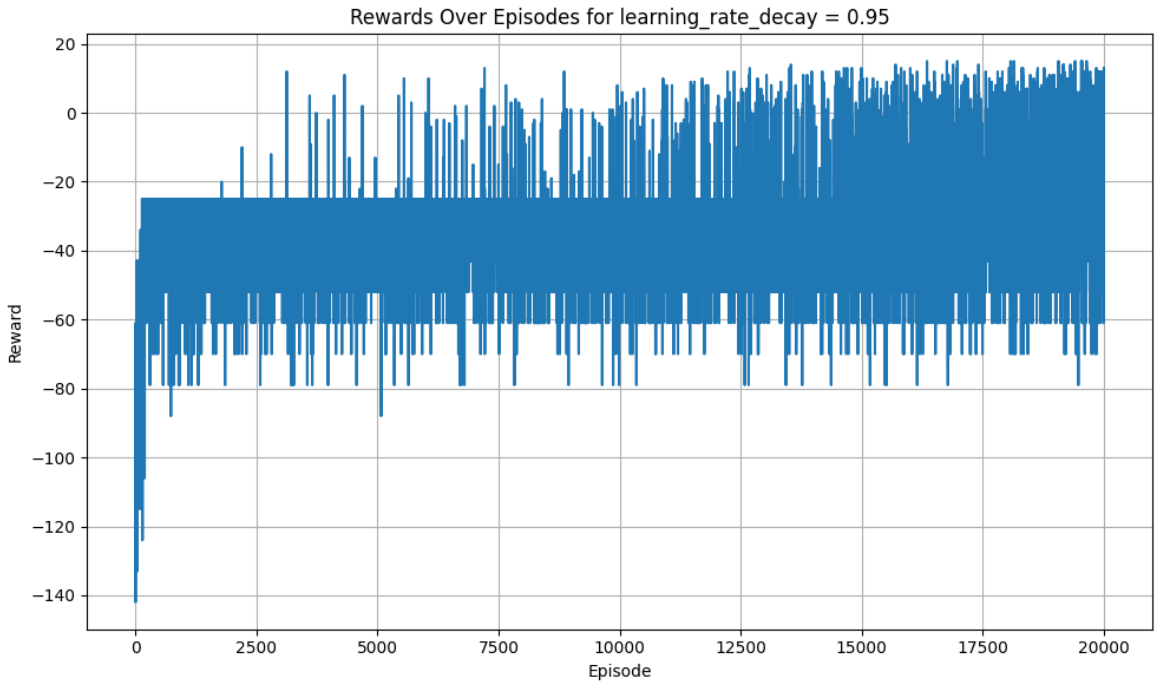

Figura 43: Recompensas con learning rate decay 0.95

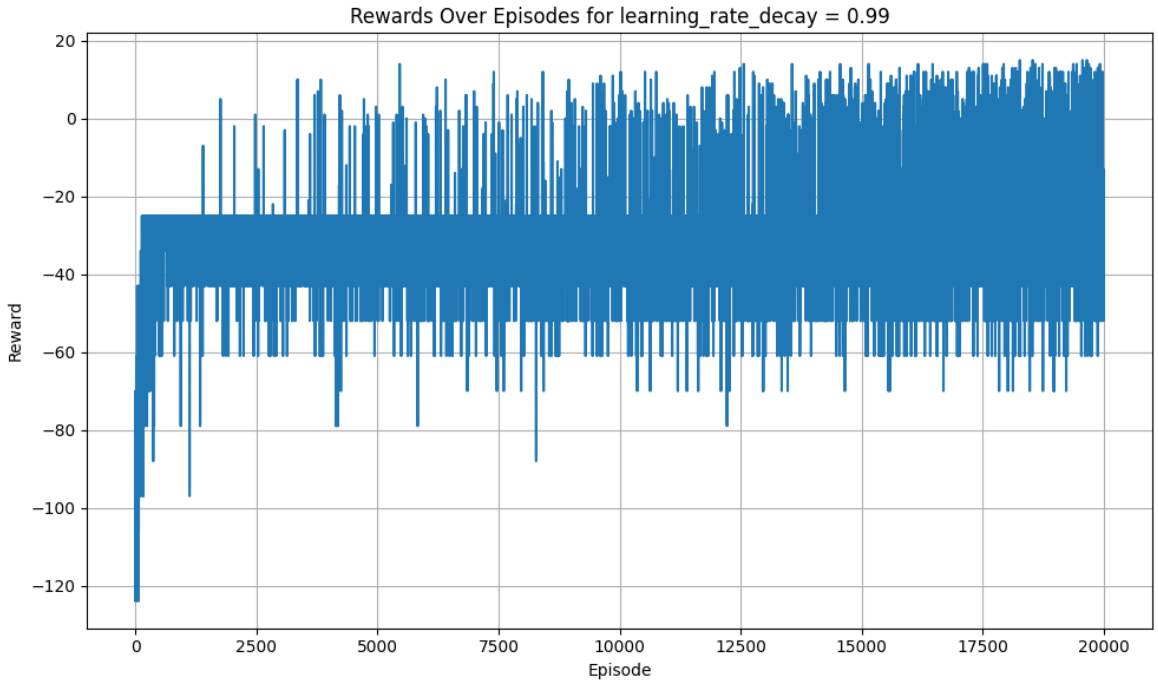

Figura 44: Recompensas con learning rate decay 0.99

En cuanto a las métricas de tiempos de entrenamiento y las recompensas dependiendo del valor de `learning_rate_decay`, se presentan las siguientes gráficas:

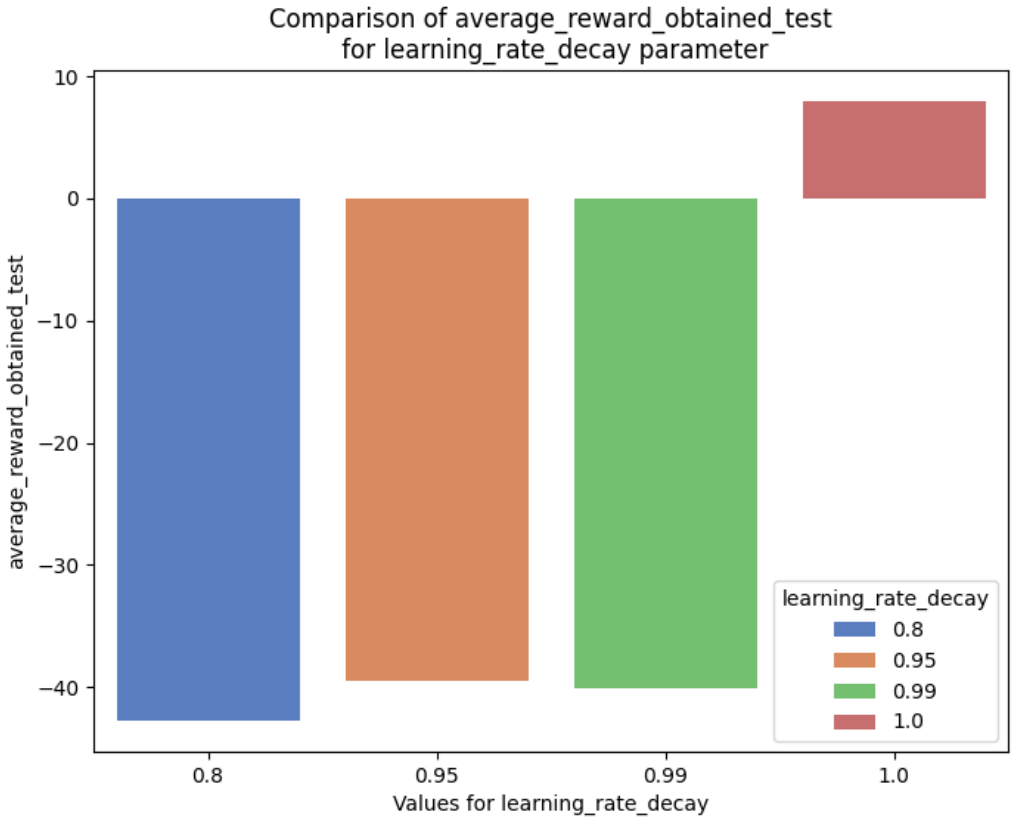

Figura 45: Recompensas promedio obtenidas en las pruebas en comparación para diferentes valores de learning rate decay

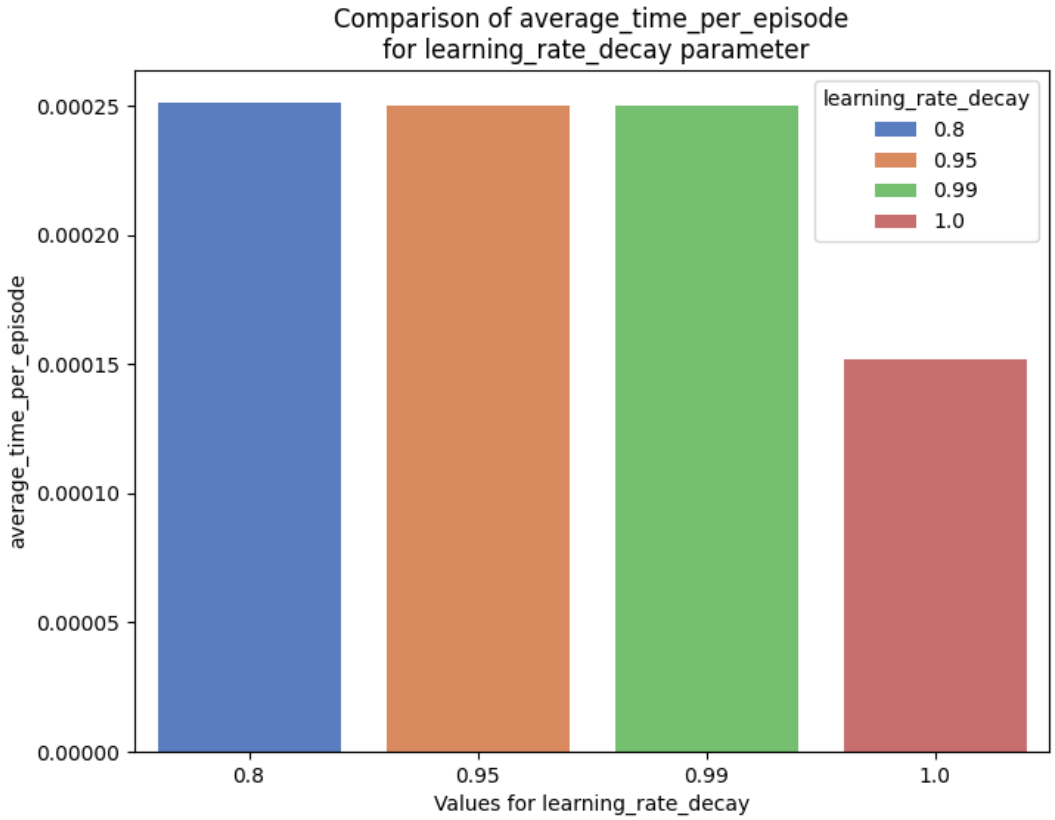

Figura 46: Tiempo promedio por episodio de entrenamiento en comparación para diferentes valores de learning rate decay

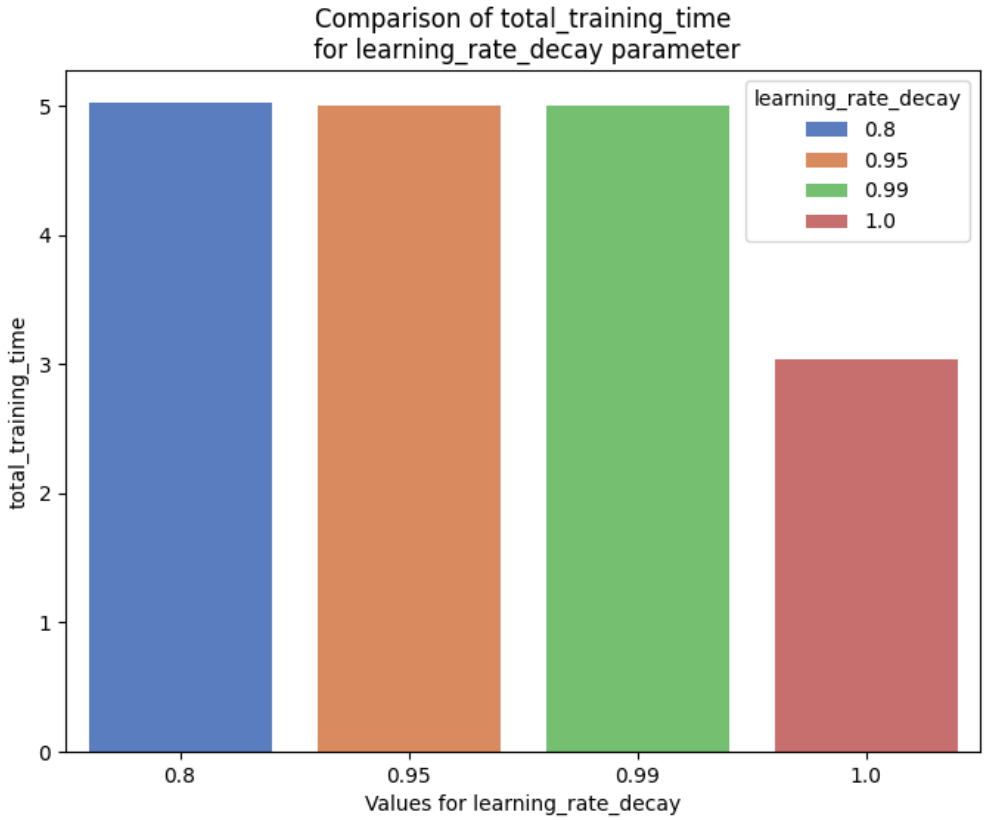

Figura 47: Tiempo total de entrenamiento en comparación para diferentes valores de learning rate decay

Vemos en las ultimas tres gráficas que obtenemos una media de recompensas mayor cuanto el decay es de 1, es decir cuando no hay decay además de que los tiempos son menores.

Resultando así la mejor opción un learning rate decay nulo.

5. **Decaimiento de Epsilon (`epsilon_decay`)**: Factores que reducen gradualmente `epsilon` a medida que el agente acumula más experiencia.

- **Hipótesis**: Un decaimiento gradual de `epsilon` puede ayudar al agente a explorar suficientemente el entorno al principio y luego explotar más su conocimiento a medida que acumula más experiencia, lo que podría mejorar su rendimiento en términos de recompensas acumuladas.

A continuación, se presentan las gráficas de las recompensas del agente con diferentes valores de `epsilon_decay`:

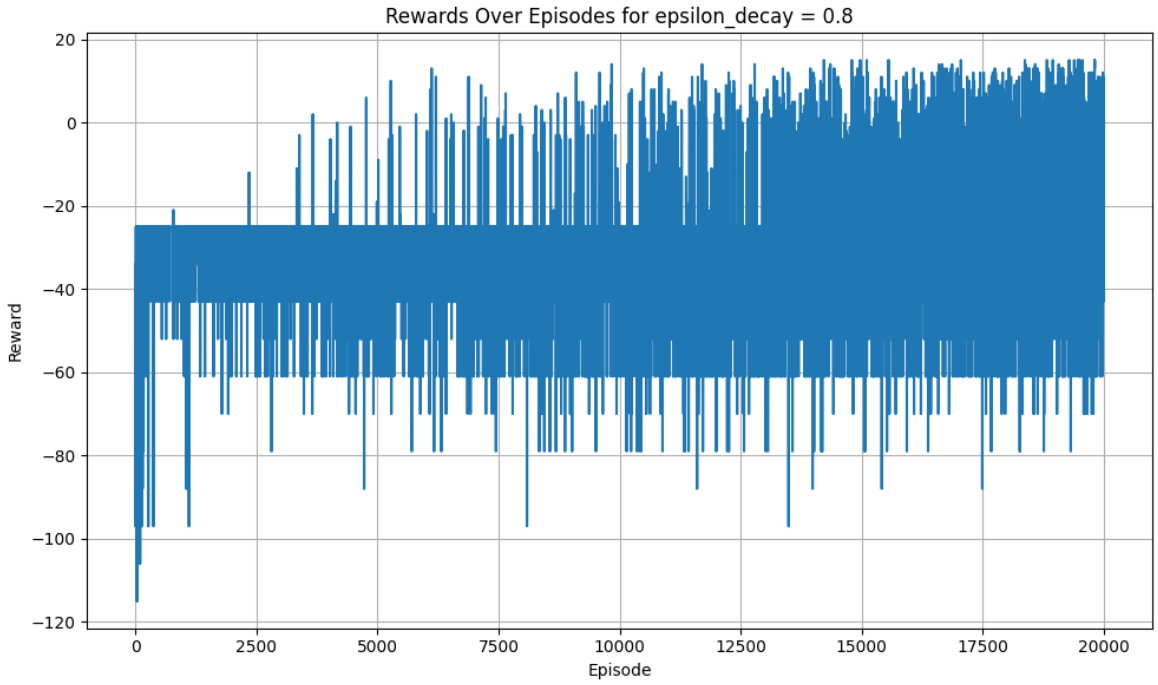

Figura 48: Recompensas con epsilon decay 0.8

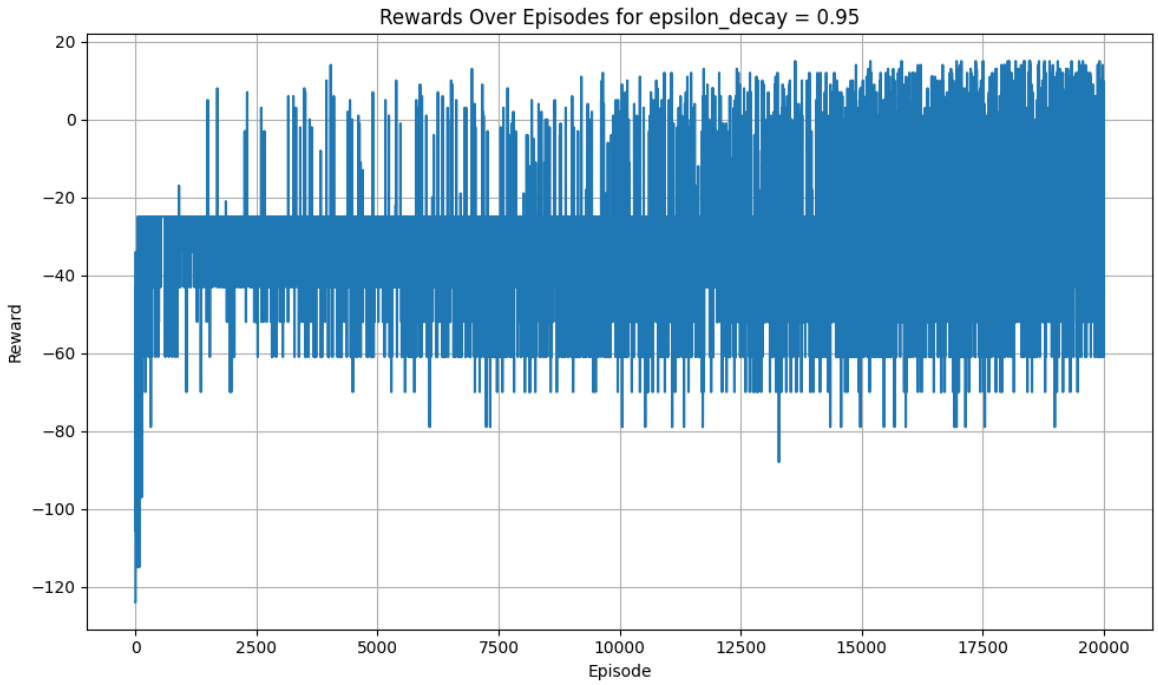

Figura 49: Recompensas con epsilon decay 0.95

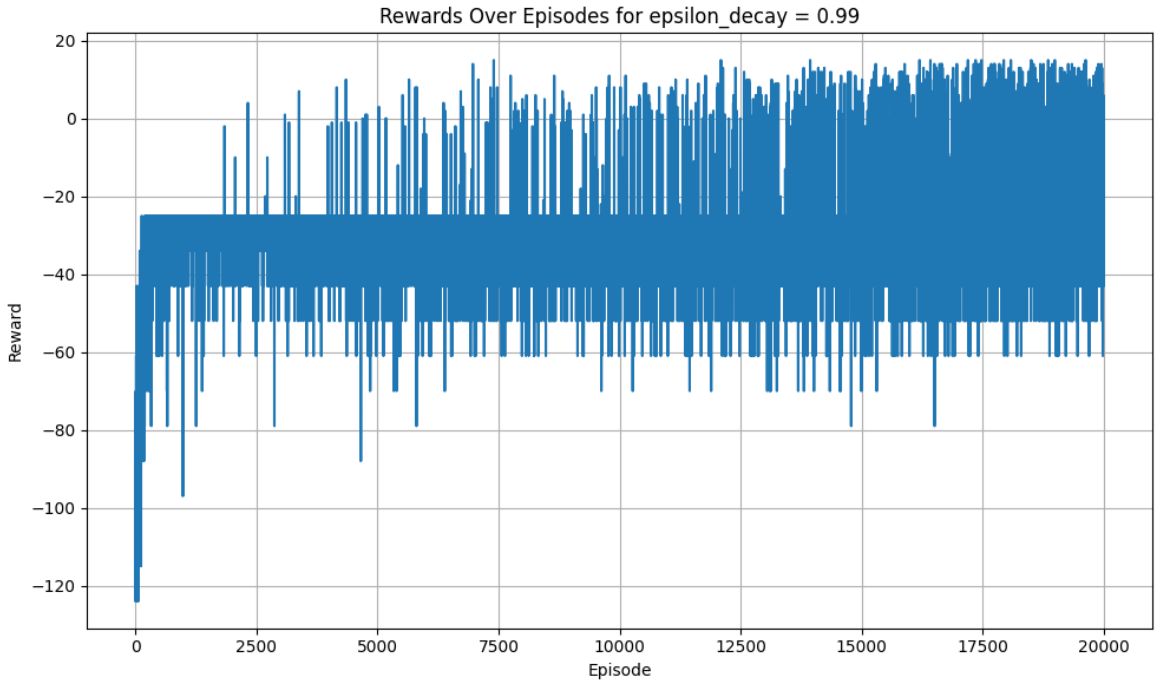

Figura 50: Recompensas con epsilon decay 0.99

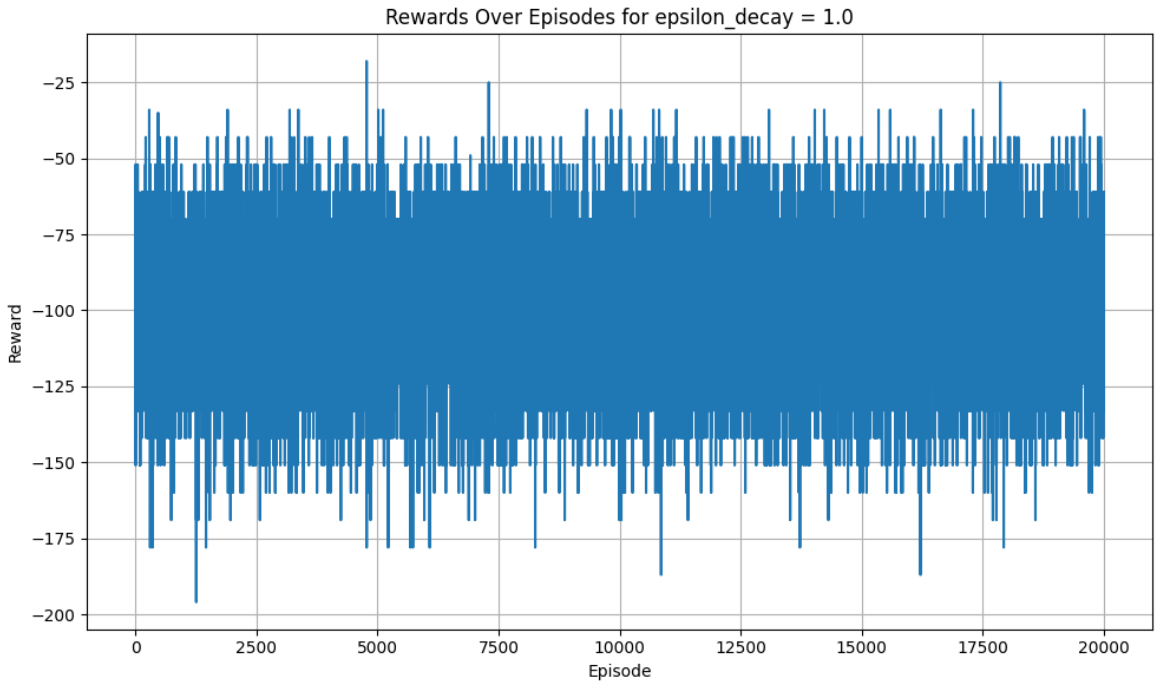

Figura 51: Recompensas con epsilon decay 1.0

En cuanto a las métricas de tiempos de entrenamiento y las recompensas dependiendo del valor de `epsilon_decay`, se presentan las siguientes gráficas:

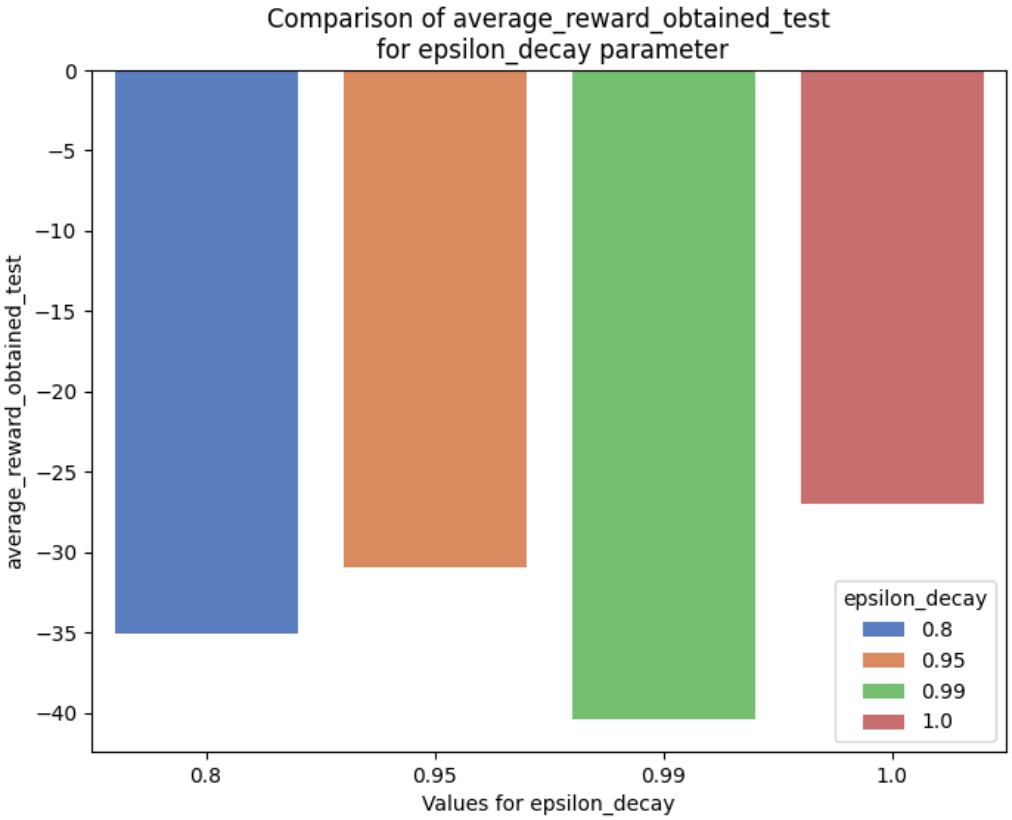

Figura 52: Recompensas promedio obtenidas en las pruebas en comparación para diferentes valores de epsilon decay

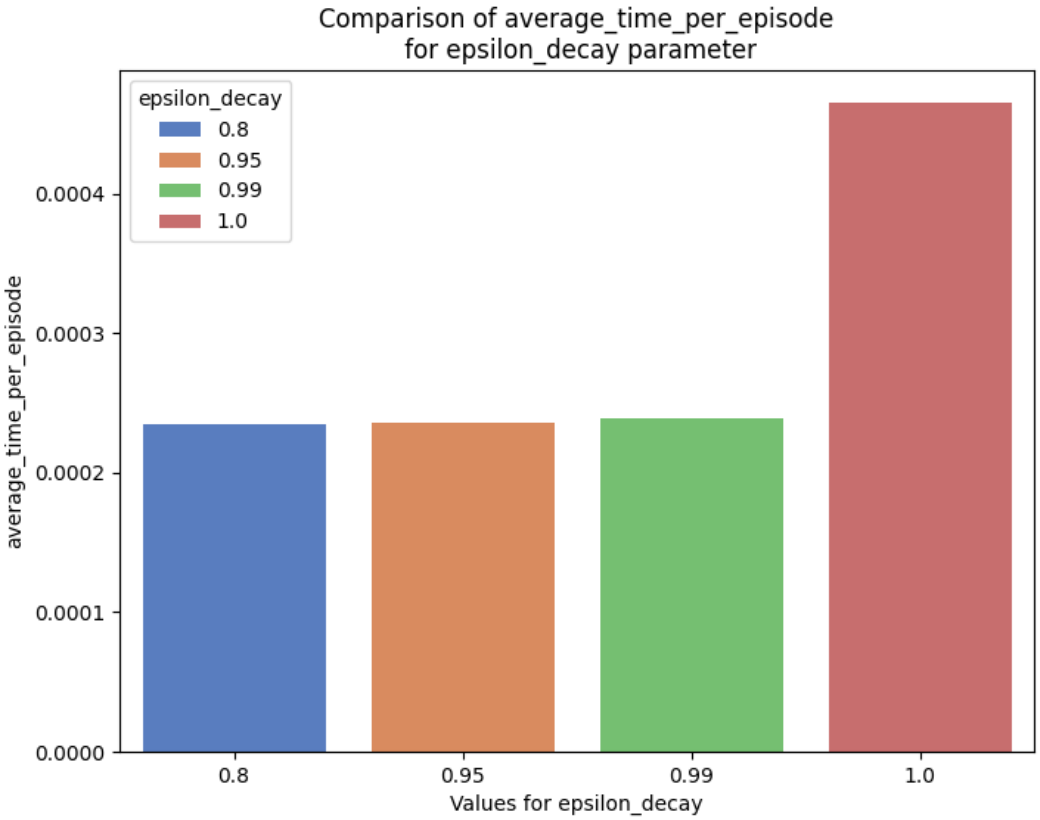

Figura 53: Tiempo promedio por episodio de entrenamiento en comparación para diferentes valores de epsilon decay

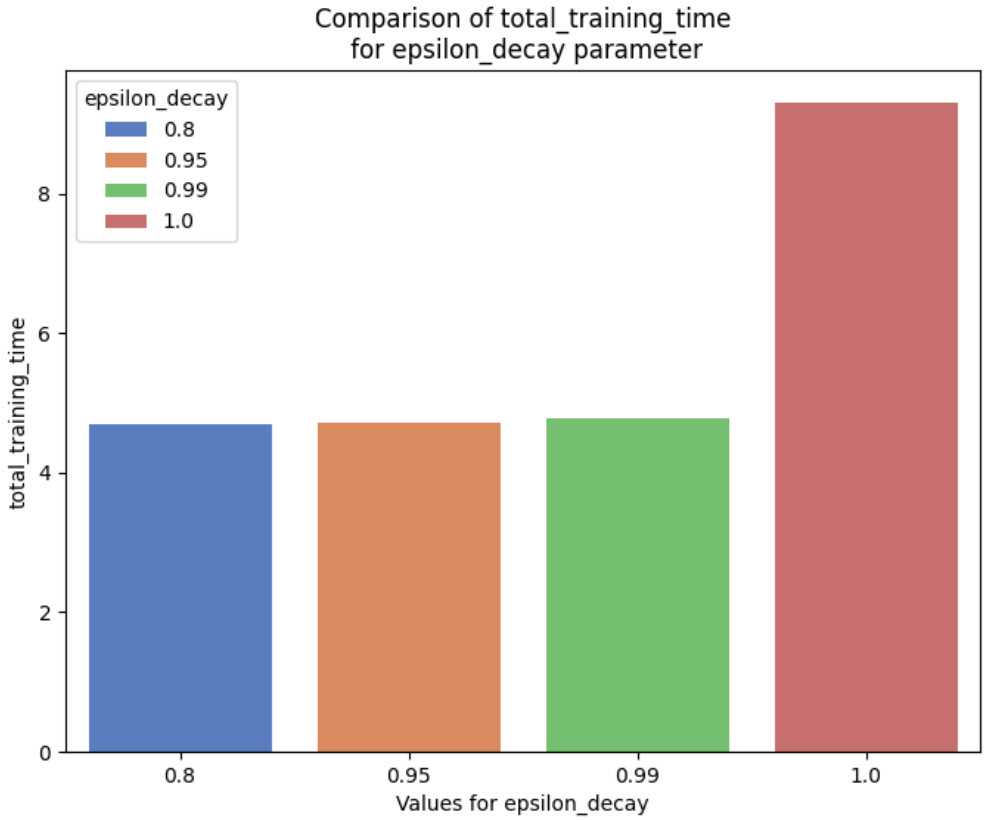

Figura 54: Tiempo total de entrenamiento en comparación para diferentes valores de epsilon decay

Vemos obtenemos la recompensa mas alta con un decay del epsilon del 0.99, es decir cuando va disminuyendo de en 1 por ciento.

Esto permite explorar mucho al inicio e ir paulatinamente explotando más y más cosa que dijimos en la hipótesis.

5. Conclusiones

Al analizar el rendimiento de los algoritmos en el entorno Taxi-v3, hemos descubierto algunas claves importantes.

Iteración por Valor

- Un valor de Gamma de 0.9 ofrece la mejor recompensa promedio. Esto equilibra bien la valoración de las recompensas futuras sin exagerarlas.
- Recomendamos un umbral de recompensa de 7.5 para evitar problemas de convergencia o tiempos de ejecución excesivos.
- Aunque un umbral más alto puede aumentar las recompensas promedio, cuidado: 8 o más puede causar ejecuciones infinitas.

Esto sugiere que, en el entorno Taxi-v3, una configuración con Gamma 0.9 y un umbral de recompensa de 7.5 es ideal para el éxito del algoritmo de Iteración por Valor.

Estimación Directa

Gamma

- Un valor intermedio de gamma (0.95) es el mejor para obtener buenas recompensas y tiempos de entrenamiento.
- Los valores extremos de gamma (0.6 y 0.99) producen resultados menos óptimos.

Número de Trayectorias

- Aumentar el número de trayectorias mejora la estabilidad del agente, lo que lleva a mejores estimaciones de valor.
- Aunque esto aumenta el tiempo de entrenamiento, la mejora en la calidad del modelo vale la pena.

Umbral de Recompensa

- Incrementar el umbral de recompensa aumenta las recompensas, pero solo hasta cierto punto.
- Más allá de un umbral de aproximadamente 8, las mejoras en las recompensas son insignificantes.

Número de Episodios

- Aumentar el número de episodios inicialmente mejora el rendimiento, pero luego no hay mejoras significativas.
- Se recomienda encontrar un equilibrio entre el número de episodios y el tiempo de entrenamiento.

Q-Learning

Al observar los hiperparámetros del Q-Learning, hemos notado lo siguiente:

- El factor de descuento (gamma) impacta significativamente el rendimiento. Valores más altos favorecen un mejor rendimiento a largo plazo, pero aumentan el tiempo de entrenamiento.
- La tasa de aprendizaje influye en la rapidez con la que el agente actualiza sus estimaciones. Valores moderados, alrededor de 0.1, parecen ser óptimos.
- El parámetro de exploración-explotación controla la cantidad de exploración que realiza el agente. Valores intermedios, como 0.5, parecen ser efectivos para equilibrar entre exploración y explotación.
- El decaimiento de la tasa de aprendizaje y el decaimiento de epsilon pueden mejorar el rendimiento a largo plazo.

En resumen, elegir los hiperparámetros adecuados es crucial para el éxito del algoritmo Q-Learning.