Aprendizaje por Refuerzo para Navegación en Laberintos con Q-learning

Andrés Castellano y Joshua Sancho

I. Introducción

El aprendizaje por refuerzo (Reinforcement Learning, RL) es una rama del aprendizaje automático que permite a un agente aprender a tomar decisiones óptimas mediante la interacción con un entorno dinámico. A través de la experiencia, el agente ajusta su política de acciones para maximizar la recompensa acumulada.

En este trabajo, se propone un enfoque basado en *Q-learning* para la navegación de un agente en laberintos discretos. El agente aprende a evitar obstáculos y alcanzar la meta mediante la actualización iterativa de una tabla de *Q-values*, que representa la utilidad esperada de cada acción en cada estado.

Se implementó un entorno simulado que permite definir laberintos con diferentes configuraciones de celdas, posiciones de inicio y meta, y paredes, junto con un módulo de visualización que facilita la observación del desempeño del agente. Este estudio ilustra cómo algoritmos de aprendizaje por refuerzo pueden aplicarse a problemas de planificación y navegación, proporcionando un marco educativo y experimental para el análisis de estrategias de decisión autónoma.

II. Metodología

II-A. Definición del entorno

El laberinto se modela como una **rejilla discreta** de tamaño $M \times N$, donde cada celda puede ser transitable o contener una pared. Se definen un **estado inicial** s_0 y un **objetivo** s_g, y el agente puede moverse en cuatro direcciones: arriba, abajo, izquierda y derecha.

II-B. Estados

El conjunto de estados $\mathcal{S}$ se define como todas las celdas válidas del laberinto:

$$\mathcal{S} = \{s = (r, c) \mid 0 \leq r < M,\ 0 \leq c < N,\ s \notin \text{paredes}\}$$

Cada estado corresponde a la posición actual del agente, representada por la esquina superior izquierda de la celda ocupada. En la *Q-table*, cada entrada $Q(s, a)$ representa el valor estimado de tomar la acción $a \in \mathcal{A}$ desde el estado $s \in \mathcal{S}$.

II-C. Acciones

El conjunto de acciones $\mathcal{A}$ se define como:

$$\mathcal{A} = \{\text{arriba}, \text{abajo}, \text{izquierda}, \text{derecha}\}$$

Cada acción a actualiza la posición del agente si el movimiento no está bloqueado por una pared o los límites del laberinto.

II-D. Recompensas

Se define la función de recompensa $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ de la siguiente manera:

$$R(s, a) = \begin{cases} +1{,}0 & \text{si } s' = s_g \\ -0{,}01 & \text{si } s' \neq s_g \end{cases}$$

donde s' es el estado resultante de aplicar la acción a en el estado s. Si el movimiento está bloqueado, el agente permanece en s y recibe la penalización de -0.01.

II-E. Algoritmo Q-learning

Se utiliza el algoritmo *Q-learning*, que actualiza iterativamente la *Q-table* según la ecuación:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \Big[r + \gamma \max_{a' \in \mathcal{A}} Q(s', a') - Q(s, a) \Big]$$

donde:

- $\alpha \in [0, 1]$ es la **tasa de aprendizaje**.
- $\gamma \in [0, 1]$ es el **factor de descuento** para recompensas futuras.
- $r = R(s, a)$ es la recompensa recibida tras ejecutar la acción a desde el estado s.
- s' es el estado resultante tras la acción a.

II-F. Política ϵ-greedy

La selección de acciones sigue una *política ϵ-greedy*:

$$a = \begin{cases} \text{acción aleatoria} & \text{con probabilidad } \epsilon \\ \arg\max_{a \in \mathcal{A}} Q(s, a) & \text{con probabilidad } 1 - \epsilon \end{cases}$$

Al inicio, ϵ es alto ($\epsilon = 1$) para favorecer la exploración, y se reduce progresivamente según un *decay* para favorecer la explotación de la política aprendida.

II-G. Visualización y evaluación

Se registran las posiciones del agente en cada episodio y se generan animaciones que muestran la evolución del aprendizaje. Además, se realiza una **validación final** usando el último checkpoint de la *Q-table* para medir la eficiencia del agente en alcanzar el objetivo.

III. Resultados

El agente fue entrenado durante 2240 episodios en un laberinto de prueba hasta que la política aprendida se estabilizó, indicando convergencia. A continuación, se presentan las métricas obtenidas al final del entrenamiento:

La Figura 1 muestra la evolución de la recompensa total por episodio durante el entrenamiento. Se observa que, a medida que avanza el entrenamiento, la recompensa promedio

Métrica	Valor
Episodios totales	2240
Recompensa total acumulada	1082.52
Recompensa promedio por episodio	0.48
Mejor recompensa en un episodio	0.76
Peor recompensa en un episodio	-34.18
Tiempo promedio por episodio	0.0004 s
Episodio más rápido	0.0001 s
Episodio más lento	0.0352 s

TABLE I

RESUMEN DEL ENTRENAMIENTO DEL AGENTE INCLUYENDO RECOMPENSAS Y TIEMPOS DE EJECUCIÓN.

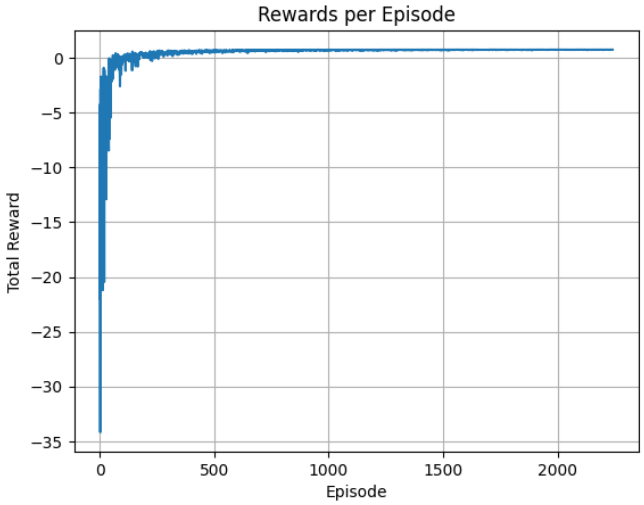

Fig. 1. Recompensa total por episodio durante el entrenamiento.

tiende a estabilizarse, indicando que el agente ha aprendido una política efectiva para navegar el laberinto.

La Figura 2 muestra el tiempo de ejecución por episodio durante el entrenamiento. Se observa que la mayoría de los episodios se ejecutan muy rápido, aunque algunos episodios presentan un tiempo significativamente mayor debido a la exploración inicial del agente.

Además, se evaluó el agente utilizando el último checkpoint en un episodio de validación. El agente logró alcanzar la meta evitando obstáculos con el mínimo número de pasos, demostrando la eficacia de la política aprendida.

El entrenamiento se realizó en un equipo con las siguientes características:

- Procesador: AMD Ryzen 7 5800X @ 4.2 GHz
- Memoria RAM: 32 GB DDR4 @ 1066 MHz
- Sistema operativo: Ubuntu 21.04 (WSL)
- Python versión: 3.11.14
- Librerías: NumPy, Matplotlib

IV. DISCUSIÓN

Los resultados muestran que el agente entrenado mediante *Q-learning* fue capaz de aprender una política efectiva para navegar el laberinto. La recompensa promedio por episodio

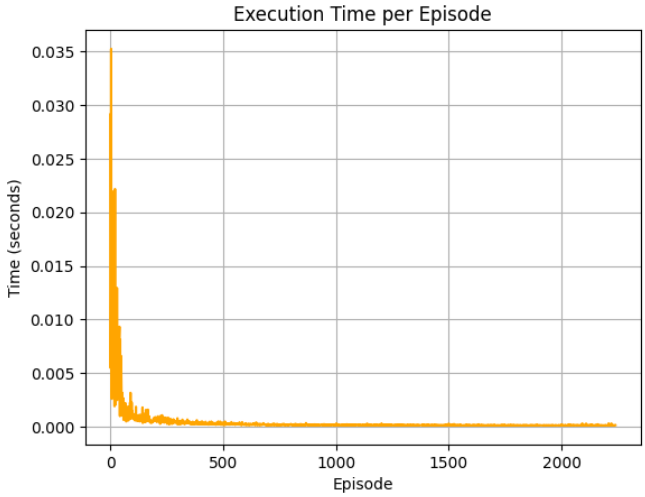

Fig. 2. Tiempo de ejecución por episodio durante el entrenamiento.

se estabilizó a lo largo de los 2240 episodios de entrenamiento, indicando que el agente alcanzó convergencia y que la *Q-table* refleja adecuadamente la utilidad de cada acción en cada estado.

Durante la validación, el agente logró alcanzar la meta evitando obstáculos y minimizando el número de pasos, confirmando la eficacia de la política aprendida, sin embargo, se observaron episodios con recompensas extremas: la mejor recompensa registrada fue 0.76 y la peor -34.18. Estas diferencias se atribuyen a la exploración inicial del agente, impulsada por un valor alto de ϵ, que ocasiona acciones aleatorias en los primeros episodios.

Los hiperparámetros del agente, como la tasa de aprendizaje α, el factor de descuento γ y el parámetro de exploración ϵ, influyeron directamente en la velocidad de convergencia y la estabilidad de la política. El uso de un decay en ϵ permitió reducir gradualmente la exploración, facilitando que el agente explotara el conocimiento adquirido y optimizara su desempeño.

Entre las limitaciones del enfoque se encuentra su dependencia del entorno específico: cualquier cambio en la configuración del laberinto (paredes, posición de inicio o meta) requeriría reentrenamiento, además, el método se adapta mejor a laberintos discretos de tamaño moderado; laberintos grandes o continuos podrían requerir aproximaciones de *Q-function*, como *Deep Q-learning*.

Finalmente, los resultados obtenidos evidencian que *Q-learning* puede aplicarse a problemas de navegación y planificación de rutas en entornos discretos. Futuras mejoras podrían incluir la incorporación de recompensas intermedias para acelerar el aprendizaje o la extensión del método a entornos más complejos mediante aproximaciones de función de valor.

APPENDIX

Repositorio del proyecto

El código completo del proyecto, así como las animaciones del agente en los laberintos, se encuentran disponibles

en el siguiente repositorio.