Comparación del desempeño de agentes inteligentes en entornos con obstáculos usando MESA y visualización en Unity

Valeria Arciga Valencia[1], **Ana Lidia Hernández Díaz**[1] **y Viviana Carrizales Luna**[1]

[a]*Instituto Tecnológico y de Estudios Superiores de Monterrey*

Carlos Astengo Noguez, Lorena Beatriz Martínez Elizalde

Resumen—Este trabajo de investigación analiza la eficiencia de distintos tipos de agentes inteligentes, incluyendo agentes reactivos, agentes basado en metas con el algoritmo A*, y agentes de aprendizaje por refuerzo con el algoritmo Q-Learning en entornos discretos con obstáculos.
Para ello, se empleó MESA como plataforma para la simulación del entorno y desarrollo de agentes, diseñando escenarios personalizados con estructuras reutilizables en formato JSON. Se registró y comparó el desempeño de los agentes en términos de pasos realizados, cantidad de celdas exploradas y capacidad adaptativa.
El análisis comparativo busca mejorar la comprensión de estrategias de toma de decisiones en agentes autónomos y explorar posibles aplicaciones en contextos como la exploración espacial dentro del curso *Diseño de agentes inteligentes*.

Keywords—*diseño de agentes inteligentes, aprendizaje supervisado, algoritmo A estrella, MESA*

1. Introducción

Un aspecto fundamental en el diseño de agentes inteligentes es la navegación autónoma en distintos entornos, especialmente en sistemas que operan sin intervención humana directa. Este trabajo se inspira en el caso de exploración espacial, donde robots como el Perseverance Rover deben adaptarse a entornos desconocidos, reconocer su entorno y tomar decisiones en tiempo real.

En ese contexto, se propone analizar la efectividad de distintas estrategias de navegación en entornos discretos simulados utilizando la librería MESA en Python. Se comparan tres enfoques: un agente reactivo, un agente basado en metas empleando el algoritmo A* y un agente de aprendizaje por refuerzo con Q-Learning.

Cada tipo de agente tiene características particulares: el agente reactivo responde localmente a los obstáculos sin planificación, el agente basado en metas optimiza los movimientos para hallar la ruta más eficiente y el agente con Q-Learning aprende a mejorar su rendimiento en función de la retroalimentación del entorno.

2. Metodología

2.1. Arquitectura y diseño preliminar de agentes

Los agentes operan en un espacio discretizado en dos dimensiones, donde cada celda de la cuadrícula representa un estado posible del agente. Las posiciones se representan mediante coordenadas (x,y), y los agentes pueden desplazarse a las celdas adyacentes solo atravesando las aristas, es decir, arriba, abajo, izquierda o derecha. El entorno está restringido a un número limitado de pasos por episodio, lo que significa que el agente debe actuar dentro de un límite de tiempo para alcanzar su objetivo.

El grid puede contener obstáculos que bloquean el camino del agente, así como una meta que el agente debe alcanzar. Según la configuración, el agente puede recibir recompensas o penalizaciones en función de sus acciones en función de su distancia a la meta. El entorno se caracteriza como estático, determinista, totalmente observable, discreto, episódico, con la posibilidad y de 1 agente (cabe notar que en algunas implementaciones se usan más agentes para la comparación de resultados).

En cuanto a los tipos de agentes a implementar, se tiene el agente reactivo que toma acciones inmediatas basadas únicamente en el estado actual del entorno, sin tener en cuenta el historial de acciones previas ni un objetivo a largo plazo.

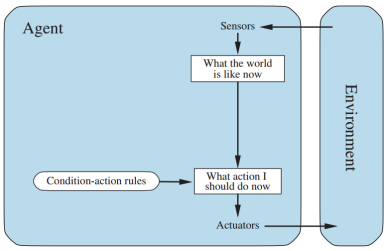

Figura 1. Agente Reactivo Simple [1]

Su arquitectura es la más simple, ya que solo necesita un modelo que reciba el estado actual del entorno como entrada y devuelva una acción sin considerar un plan o memoria de acciones pasadas.

El agente con la implementación del algoritmo A*, tiene una meta definida, por lo cual está diseñado para maximizar las probabilidades de alcanzar ese objetivo. A diferencia del agente reactivo, este tipo de agente no actúa de manera aleatoria, sino que toma decisiones informadas para acercarse a la meta. Su arquitectura incluye una representación del estado del entorno y un módulo de toma de decisiones basado en el cálculo de la distancia o la proximidad hacia la meta.

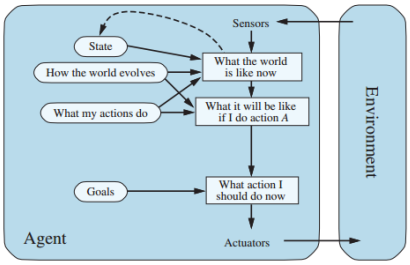

Figura 2. Agente Basado en Metas [1]

Por otro lado, Q-Learning es un enfoque dentro del aprendizaje automático que permite a un modelo mejorar su desempeño a través de múltiples iteraciones, aprendiendo a tomar decisiones acertadas con el tiempo.

Este método sigue un enfoque de aprendizaje por refuerzo no basado en políticas, lo que significa que el agente puede desarrollar una estrategia óptima en función del estado actual y ajustarla conforme obtiene más experiencia para maximizar su beneficio. Para calcular el resultado, se emplea la Ecuación de Bellman 1, una fórmula recursiva utilizada para la toma de decisiones óptimas. [2]

El modelo Q-Learning se basa en experiencias de ensayo y error, esto implica que el agente será basado en el aprendizaje. Es un modelo de aprendizaje adaptativo que mejora las decisiones del agente.

Esto, a diferencia del agente reactivo simple aleatorio, que no aprende ni optimiza su comportamiento, y A*, que sigue una estrategia predefinida sin necesidad de aprendizaje.

2.2. Desarrollo y diseño del entorno

Para construir el mundo se determinó en un archivo tipo .JSON las proporciones del entorno, tanto el punto de salida como la meta deseada y los obstáculos. Se siguió la siguiente estructura:

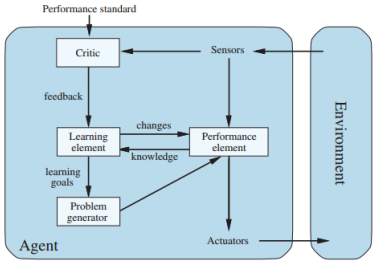

Figura 3. Agente Basado en Aprendizaje [1]

```
{
    "width": 0,
    "height": 0,
    "start": [0, 0],
    "goal": [0, 0],
    "obstacles": [
        [0,0]
    ]
}
```

Código 1. Ejemplo de configuración en JSON

2.2.1. Escenario 1: Escenario simple

START	1,0	2,0	3,0	4,0
0,1	1,1	2,1	3,1	4,1
0,2	1,2	2,2	3,2	4,2
0,3	1,3	2,3	3,3	4,3
0,4	1,4	2,4	3,4	GOAL

Figura 4. Arquitectura del escenario 1

El primer escenario fue diseñado para evaluar el desempeño de cada agente en un entorno básico. Se utilizó un tablero de 5x5 unidades sin obstáculos, permitiendo un análisis del comportamiento del agente en condiciones óptimas.

2.2.2. Visualización del escenario 1 en Unity

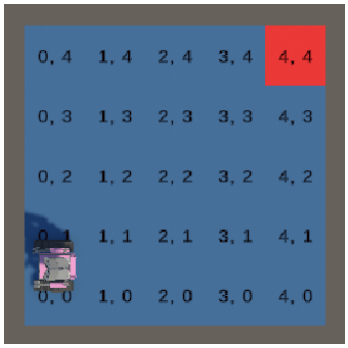

Figura 5. Visualización del escenario 1 con agente

La implementación del escenario fue realizada con un GridManager en donde permitía el ajuste del tamaño de la cuadrícula, y en donde se señala la celda que corresponde al objetivo del agente.

Es importante mencionar que la implementación de estos escenarios no tienen relación con la ejecución de los agentes inteligentes, sino que son una simulación de los resultados obtenidos.

2.2.3. Escenario 2: Escenario con obstáculos aleatorios

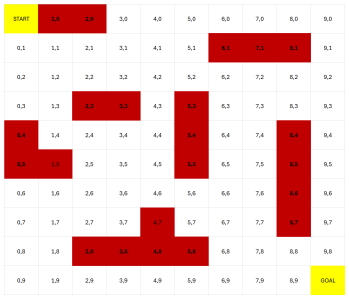

Figura 6. Arquitectura del escenario 2

En el segundo escenario, se incorporaron obstáculos de manera aleatoria, lo que generó múltiples rutas posibles hacia la meta. Esto permitió evaluar la capacidad del agente para tomar decisiones y adaptarse a distintas trayectorias.

2.2.4. Visualización del escenario 2 en Unity

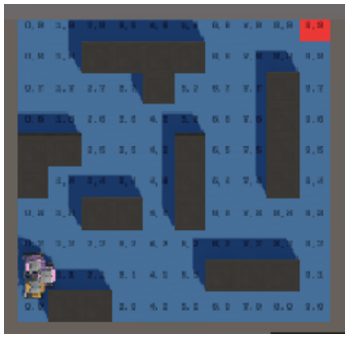

Figura 7. Visualización del escenario 2 con agente

En la segunda implementación del escenario, se hizo uso de elementos extra para la representación de los obstáculos de una forma visual, se mantiene la selección del objetivo con otro color de casilla.

2.2.5. Escenario 3: Escenario con obstáculos en diagonal, con una única salida

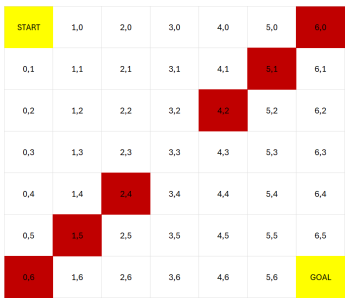

Figura 8. Arquitectura del escenario 3

En el tercer escenario, los obstáculos fueron colocados en diagonal, dejando solo una casilla libre para avanzar. Se espera que el agente perciba esta restricción y determine la única ruta viable hacia la meta.

2.2.6. Visualización del escenario 3 en Unity

Siguiendo a la figura 8, el escenario en unity fue planteado con la colocación de los objetos de forma diagonal, manteniendo la señalización del objetivo.

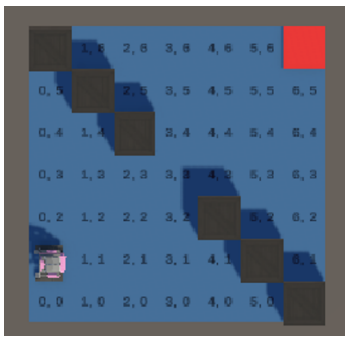

Figura 9. Visualización del escenario 3 con agente

3. Diseño de agentes implementación

3.1. Agente Reactivo

Para la implementación del agente reactivo, se desarrolló un modelo basado en reglas simples de movimiento dentro de un entorno discreto representado con la cuadrícula. El agente fue denominado "Walle reactivo", toma decisiones inmediatas sin planificación, basándose únicamente en la percepción de su entorno inmediato.

El agente cuenta con una estructura sencilla compuesta por una posición inicial, un objetivo y un conjunto de reglas que determinan su movimiento. Su lógica de navegación se basa en evaluar las posiciones adyacentes y seleccionar aleatoriamente un movimiento válido que no implique colisión con obstáculos.

Para cada paso, el agente verifica si ha alcanzado su objetivo. En caso contrario, se generan las posibles direcciones en las que puede desplazarse dentro de los límites de la cuadrícula. Se filtran aquellas posiciones ocupadas por obstáculos, y entre las opciones restantes, se elige una dirección de manera aleatoria.

La actualización de la posición se realiza con la función move agent(), garantizando que el agente se desplace dentro de la cuadrícula sin pasar sobre los obstáculos. Se registra el historial de posiciones recorridas para análisis posterior.

El entorno de simulación se implementa con la clase GridModel, que gestiona la cuadrícula y la interacción entre el agente y el espacio, recibe la información del escenario. El modelo inicializa la cuadrícula con dimensiones específicas, coloca obstáculos en posiciones predefinidas y posiciona al agente en su punto de inicio.

El proceso de simulación avanza ejecutando iterativamente la función step(), que evalúa y actualiza la posición del agente en cada iteración hasta que este alcanza la meta.

Para evaluar el desempeño del agente, se registran las trayectorias recorridas en un archivo JSON. La información almacenada incluye la posición inicial del agente y las coordenadas visitadas en su trayectoria. Estas métricas permiten analizar la eficiencia del agente en distintos escenarios con diferentes configuraciones de obstáculos, además de permitir la implementación en el entorno visual de Unity.

3.2. Agente Basado en metas

El agente con la implementación del algoritmo A*, tiene una meta definida, por lo cual está diseñado para maximizar las probabilidades de alcanzar ese objetivo. A diferencia del agente reactivo, este tipo de agente no actúa de manera aleatoria, sino que toma decisiones informadas para acercarse a la meta. Su arquitectura incluye una representación del estado del entorno y un módulo de toma de decisiones basado en el cálculo de la distancia o la proximidad hacia la meta.

Opera por medio de una función heurística que se encarga de calcular el costo de cada celda posible a explorar, combinando este valor con el costo acumulado desde el punto de inicio. Esta función heurística, comúnmente denotada como $h(n)$, estima la distancia restante hasta la meta, mientras que el costo acumulado, $g(n)$, representa el costo real del camino recorrido hasta el nodo actual.

3.3. Agente que aprende

Para la implementación del agente basado en Deep Q-Learning (DQL), se utilizó una arquitectura de red neuronal para aproximar la función de valor Q. La arquitectura de la red neuronal utilizada consta de tres capas totalmente conectadas. La primera capa recibe como entrada el estado del entorno y lo proyecta a una capa oculta de 64 neuronas. Posteriormente, una segunda capa oculta de 64 neuronas procesa la información, y finalmente, la capa de salida genera los valores Q para cada acción posible en el entorno. Se emplea la función de activación ReLU en las capas ocultas para mejorar la capacidad de aprendizaje del modelo.

Para entrenar la red, se almacenan experiencias en una memoria de repetición, la cual permite seleccionar lotes de experiencias de manera aleatoria para evitar la correlación entre muestras consecutivas. Durante el proceso de actualización, se extrae un minibatch de experiencias de la memoria, y para cada experiencia se calcula el valor objetivo de Q utilizando la red objetivo. La actualización de los pesos de la red principal se realiza mediante la minimización del error cuadrático medio entre los valores Q predichos por la red principal y los valores objetivo calculados con la red objetivo.

Para garantizar la estabilidad en el entrenamiento, la red objetivo se actualiza cada cierto número de iteraciones mediante una actualización suave, donde los parámetros de la red objetivo se modifican de acuerdo con la siguiente ecuación:

$\theta_{\text{target}} = \tau\theta_{\text{model}} + (1-\tau)\theta_{\text{target}}$ donde θ_{model} representa los pesos de la red principal, θ_{target} representa los pesos de la red objetivo y τ es un parámetro de interpolación que controla la tasa de actualización. En este caso, se utilizó un valor de $\tau = 0.01$ para evitar cambios bruscos en la red objetivo.

Finalmente, el proceso de entrenamiento se repite durante múltiples episodios, ajustando los pesos de la red neuronal hasta que el agente logre mejorar su desempeño en la tarea de navegación y toma de decisiones dentro del entorno definido.

4. Resultados y análisis comparativo

4.1. Escenario 1

Agente	Observación 1	Observación 2	Observación 3	Promedio pasos
Reactivo simple aleatorio	60	54	64	59.33
Algoritmo A*	8	8	8	8
DQL-Learning	10	10	8	9.33

Agente	Observación 1	Observación 2	Observación 3	Promedio de celdas exploradas
Reactivo simple aleatorio	20	17	22	19.67
Algoritmo A*	25	25	25	25
DQL-Learning	23	22	23	22.66

4.2. Escenario 2

Agente	Observación 1	Observación 2	Observación 3	Promedio pasos
Reactivo simple aleatorio	134	146	258	179.33
Algoritmo A*	18	18	18	18
DQL-Learning	18	26	22	22

Agente	Observación 1	Observación 2	Observación 3	Promedio de celdas exploradas
Reactivo simple aleatorio	49	43	53	48.3
Algoritmo A*	58	58	58	58
DQL-Learning	59	63	75	65.66

4.3. Escenario 3

Agente	Observación 1	Observación 2	Observación 3	Promedio pasos
Reactivo simple aleatorio	46	576	22	214.67
Algoritmo A*	12	12	12	12
DQL-Learning	36	14	12	20.66

Agente	Observación 1	Observación 2	Observación 3	Promedio de celdas exploradas
Reactivo simple aleatorio	18	35	19	24
Algoritmo A*	37	37	37	37
DQL-Learning	44	43	47	44.6

En la Figura 10, 11 y 12, se muestra la recompensa total obtenida por episodio durante el entrenamiento del agente DQL en los distintos escenarios. Se puede observar cómo el agente aprende progresivamente a maximizar la recompensa, ajustando sus estrategias de acción con base en la retroalimentación recibida.

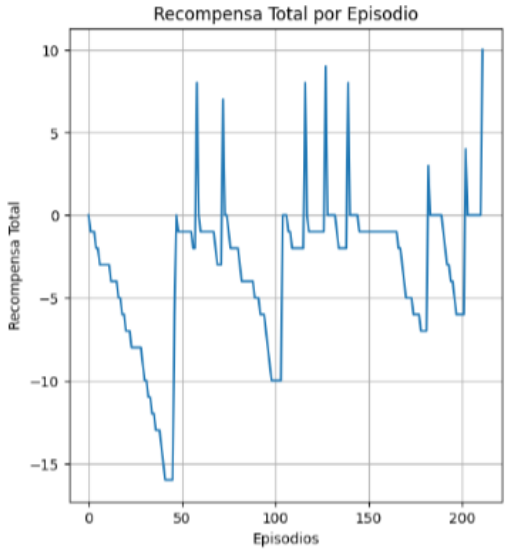

Figura 10. Recompensa total por episodio en entrenamiento con escenario 5x5 tercera simulación, con 10 episodios.

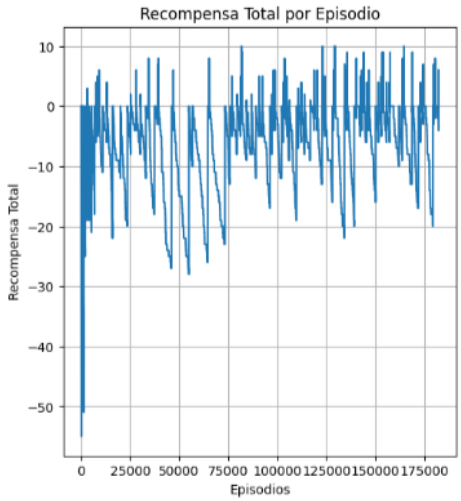

Figura 11: Recompensa total por episodio en entrenamiento con escenario 10x10 segunda simulación, con 100 episodios

5. Principales Hallazgos y Reflexiones

Tras evaluar el comportamiento del **Agente Reactivo Simple Aleatorio**, se puede concluir que este tipo de comportamiento puede resultar poco práctico si lo que queremos es minimizar el número de pasos o llegar a la meta en un corto periodo de tiempo. No obstante, puede ser altamente útil para casos donde se quiera evitar la predictibilidad frente a otros agentes, por lo cual si se está en un contexto multiagente es una solución bastante racional.

De la misma forma, este tipo de comportamiento es ideal para no caer en algún tipo de bucle, por ejemplo, si el agente fuese una aspiradora que percibe si su entorno está *Limpio*, puede elegir entre izquierda o derecha para limpiar la siguiente casilla y en dado caso que estuviera sucia, la limpiaría y completaría su tarea. En este contexto, un agente reactivo simple que actúa de forma aleatoria podría superar en desempeño a uno determinista ya que evita esos ciclos repetitivos terminando completando la tarea de forma satisfactoria [1].

Por otro lado, el agente modelado con el **Algoritmo A*** resultó ser un método más efectivo para esta prueba debido a la naturaleza del algoritmo, que siempre busca encontrar la ruta más corta. Al evaluar el entorno y las opciones disponibles, el algoritmo A* utiliza una

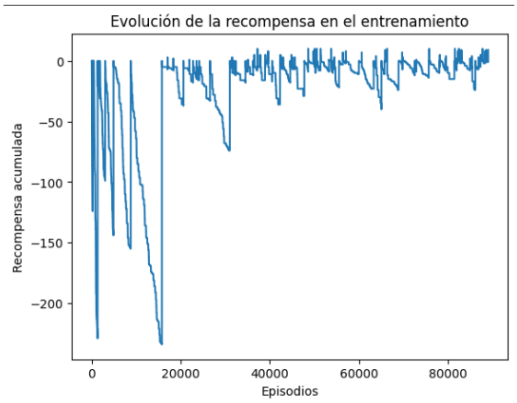

Figura 12: Recompensa total por episodio en entrenamiento con escenario 7x7 segunda simulación, con 100 episodios

combinación de costos acumulados y estimaciones hacia el objetivo, asegurando así que el agente tome el camino más eficiente posible.

En el escenario 1, el agente se desplazó en forma de L. Este comportamiento puede justificarse por el uso del movimiento en las 4 direcciones cardinales, lo que limita las opciones de desplazamiento del agente. Además, al ser la distancia Manhattan la medida utilizada, el agente opta por seguir un camino recto en la medida de lo posible, lo que da como resultado un trayecto en forma de L.

En términos de desempeño, el Agente con DQL muestra una flexibilidad que puede ser más eficiente en escenarios complejos donde las condiciones no son estáticas. Aunque su proceso de aprendizaje puede ser más lento en comparación con los métodos deterministas, la capacidad de adaptarse y aprender de las experiencias previas le otorgan una ventaja significativa en situaciones más dinámicas. Sin embargo en esta implementación se tuvo problemas para ajustar la red de acuerdo a los ambientes presentados, dando como resultado desempeños iguales o menores a los obtenidos con el algoritmo A* por lo que si bien, los agentes que aprenden son de utilidad en problemas complejos, especialmente en ambientes no discretizados, en problemas simples como en los ambientes presentados, la implementación de DQL no es oportuna.

6. Aplicaciones potenciales

La implementación de agentes inteligentes para la resolución de problemas en distintos entornos los convierte en una herramienta útil en diferentes dominios. Si bien los agentes simples se encuentran en la base de la jerarquía de inteligencia artificial, son extremadamente útiles debido a su arquitectura sencilla, que les permite operar de manera rápida y eficiente en entornos controlados y bien definidos. Aunque carecen de flexibilidad para enfrentarse a entornos dinámicos, su eficiencia y bajo coste computacional hacen que sean imprescindibles en aplicaciones como la automatización de procesos básicos, control de robots en entornos controlados y tareas con bajo nivel de incertidumbre.

En el caso de los sistemas de navegación, los agentes basados en metas juegan un papel crucial al permitir que un sistema, como un robot o un vehículo autónomo, alcance un objetivo específico de manera efectiva. En el ámbito de la navegación, esto es vital para garantizar que los sistemas puedan tomar decisiones informadas y adaptar su comportamiento en función de variaciones en el entorno, como obstáculos o cambios de ruta. La implementación de Deep Q-Learning (DQL) en vehículos autónomos lleva la navegación de estos sistemas a un nivel completamente diferente.

DQL, es una técnica de aprendizaje por refuerzo que utiliza redes neuronales profundas para aproximar la función de valor, y permite que un vehículo autónomo aprenda a tomar decisiones complejas basadas en la retroalimentación de su entorno. A través de la interacción constante con el entorno, el agente aprende no solo a reaccionar ante

obstáculos y situaciones, sino también a optimizar su comportamiento para maximizar la seguridad y eficiencia de la conducción a largo plazo. Este enfoque es especialmente útil en entornos dinámicos y no deterministas, como el tráfico urbano, donde los vehículos deben tomar decisiones complejas, como cambiar de carril, frenar o acelerar en función de una multitud de variables.

Además, el DQL es de vital importancia para aquellos ambientes que no se pueden discretizar fácilmente, por lo que la implementación de enfoques como Q-learning se convierte insostenible. En estos casos, la implementación de Deep Q-Learning (DQL) se vuelve crucial, ya que utiliza redes neuronales profundas para aproximar la función de valor en lugar de depender de una tabla explícita. Esto permite a los agentes aprender en entornos continuos y de alta dimensionalidad sin la necesidad de discretizar todos los posibles estados y acciones, lo que sería inviable tanto en términos de tiempo de cómputo como de almacenamiento.

En el caso de vehículos autónomos que deben operar en un entorno dinámico y continuo, como una ciudad con tráfico, el número de posibles combinaciones de estados y acciones se vuelve extremadamente grande y no se puede gestionar de manera efectiva con métodos tradicionales como el Q-Learning tabular. En este tipo de entorno, las redes neuronales profundas se utilizan para aproximar la función de valor y las políticas de acción, permitiendo que el agente maneje decisiones en espacios continuos (como coordenadas espaciales, velocidades, etc.) y optimice sus acciones de forma generalizada.

7. Conclusiones

Este estudio ha permitido evaluar y comparar el desempeño de diferentes estrategias de navegación en agentes inteligentes para entornos discretos con obstáculos. Los resultados dan muestra de que, aunque el agente reactivo ofrece ventajas en términos de simplicidad en el código y evita la predecibilidad en escenarios multiagente, su eficiencia se ve comprometida al incrementar el número de pasos requeridos para alcanzar la meta y al aumentar la complejidad de los escenarios.

Por su parte, el agente basado en metas mediante el algoritmo A* demostró ser el más eficaz para determinar la ruta óptima en cada uno de los escenarios, minimizando tanto el número de pasos como las celdas exploradas. Esto lo consolida como la mejor opción en entornos controlados como los que se han empleado en este estudio, donde la precisión y la eficiencia son primordiales.

El enfoque basado en Deep Q-Learning posee el potencial para aprender y adaptarse en entornos complejos, sin embargo, presentó desafíos en su ajuste para los escenarios discretos evaluados. En este contexto, su desempeño fue comparable o incluso inferior al del algoritmo A*, lo que sugiere la necesidad de optimizaciones adicionales o la integración de otras técnicas con aprendizaje por refuerzo.

En conclusión, la selección de la estrategia de navegación depende de las características específicas del entorno y los objetivos del sistema. Mientras que el algoritmo A* resulta altamente efectivo en escenarios estructurados, el aprendizaje por refuerzo ofrece perspectivas interesantes para entornos dinámicos y de mayor complejidad. Resulta de gran importancia la comparación de resultados para la selección de un agente adecuado. En otros estudios similares a este se recomienda explorar soluciones híbridas o con más técnicas que potencien la eficiencia de los agentes inteligentes, integrando lo mejor de ambos enfoques para abordar retos en contextos variados.

Referencias

[1] S. J. Russell y P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th. Pearson, 2020.

[2] S. M. Kerner. «What is Q-learning?» Accessed: 2025-03-13. (nov. de 2024), dirección: https : / / www . techtarget . com / searchenterpriseai/definition/Q-learning.

Anexos

.1. Ecuación de Bellman

La ecuación utilizada en el modelo Q-Learning para la toma de decisiones óptimas es la siguiente:

$$Q(s,a) = Q(s,a) + \alpha \cdot (r + \gamma \cdot \max(Q(s',a')) - Q(s,a)) \quad (1)$$

Esta ecuación permite actualizar el valor de $Q(s,a)$ con base en la recompensa obtenida y la mejor estimación futura, facilitando el aprendizaje del agente a través de ensayo y error.

.2. Repositorio de GitHub

El código fuente del proyecto integrador está disponible en el siguiente repositorio de GitHub:

https://github.com/anahedi/Proyecto-Integrador-Dise-o-de-Agentes-Inteligentes.

Autores: Valeria Arciga Valencia, Viviana Carrizales Luna y Ana Lidia Hernández Díaz.

.3. Escena 1

A continuación se muestra la configuración del primer escenario:

```
{
    "width": 5,
    "height": 5,
    "start": [0, 0],
    "goal": [4, 4],
    "obstacles": []
}
```

Código 2. Configuración del Escenario 1

.4. Escena 2

A continuación se muestra la configuración del segundo escenario:

```
{
    "width": 10,
    "height": 10,
    "start": [0, 0],
    "goal": [9, 9],
    "obstacles": [
        [1, 0], [2, 0], [6, 1], [7, 1], [8, 1],
        [2, 3], [3, 3], [5, 3], [0, 4], [5, 4],
        [8, 4],[0, 5],[5, 5],[8, 5],[8, 6],[8, 7],
        [2, 8],[3, 8],[4, 8],[5, 8],[1, 5],[4, 7]
    ]
}
```

Código 3. Configuración del Escenario 2

.5. Escena 3

A continuación se muestra la configuración del tercer escenario:

```
{
    "width": 7,
    "height": 7,
    "start": [0, 0],
    "goal": [6, 6],
    "obstacles": [
        [0, 6], [1, 5], [2, 4], [4, 2],
        [5, 1], [6, 0]
    ]
}
```

Código 4. Configuración del Escenario 3

.6. Prompts de AI

https://docs.google.com/document/d/1kVfAdNw3vEc3nfVA90Ayps3jiRpayp4yinLNtC1EyEU/edit?usp=sharing.