Comparación de tipos de Gráficas para análisis de características en Machine Learning

Universidad Santo Tomás
Facultad de Ingeniería Electrónica
EstudianteS: Cative Cristian - Nivia Julian
Asignatura: Introducción a la Inteligencia Artificial
Docente: Mg. Diego Alejandro Barragán Vargas

Abstract

This laboratory report presents the implementation and analysis of three classical problems in Artificial Intelligence: the 8-puzzle, simple state-action representations (lamp, virtual pet, treasure search), and maze exploration. The objective of the laboratory was to introduce the concept of state spaces, available actions, and goal states, as well as to apply search algorithms such as Breadth-First Search (BFS). Each problem was modeled through states and transitions, and solutions were obtained step by step. Additionally, a public GitHub repository was used to manage code versions, track progress, and document continuous improvements. The report highlights the importance of representing problems as state spaces, the role of search strategies in finding solutions, and the impact of obstacles in planning. Results demonstrate that even simple problems can illustrate fundamental concepts that serve as a basis for more advanced AI methods.

Resumen

Este informe de laboratorio presenta la implementación y análisis de tres problemas clásicos en Inteligencia Artificial: el 8-puzzle, representaciones simples de espacio de estados y acciones (lámpara, mascota virtual, búsqueda del tesoro) y la exploración de un laberinto. El objetivo del laboratorio fue introducir el concepto de espacio de estados, acciones disponibles y estado meta, además de aplicar algoritmos de búsqueda como la Búsqueda en Anchura (BFS). Cada problema fue modelado a través de estados y transiciones, obteniendo soluciones paso a paso. Adicionalmente, se empleó un repositorio en GitHub para gestionar versiones del código, registrar avances y documentar mejoras continuas. El informe resalta la importancia de representar los problemas como espacios de estados, el papel de las estrategias de búsqueda en la obtención de soluciones y el efecto de los obstáculos en la planificación. Los resultados demuestran que incluso problemas sencillos permiten ilustrar conceptos fundamentales que sirven como base para métodos más avanzados de IA.

I. Introducción

La representación de problemas mediante espacios de estados constituye un enfoque fundamental en la Inteligencia Artificial (IA). Un espacio de estados se define como el conjunto de todas las configuraciones posibles de un sistema, junto con las acciones que permiten transitar de un estado a otro. Este marco conceptual posibilita el análisis sistemático de problemas de búsqueda, planificación y toma de decisiones.

El presente laboratorio se estructuró en tres puntos que ilustran progresivamente la complejidad de la representación de problemas. En primer lugar, se abordó el *8-puzzle*, un clásico problema de búsqueda en el cual se debe transformar un estado inicial en un estado meta a través de movimientos válidos del espacio vacío. Posteriormente, se implementaron ejemplos sencillos como la lámpara, la mascota virtual y la búsqueda de un tesoro en una cuadrícula, con el fin de reforzar la comprensión de estados, acciones y metas en entornos más reducidos. Finalmente, se desarrolló un laberinto en un mundo 2×2 y posteriormente en un mundo 3×3, incluyendo obstáculos y la construcción de tablas de transición.

La metodología empleada se centró en el uso de algoritmos de búsqueda, principalmente la Búsqueda en Anchura (BFS), que garantiza la obtención de soluciones óptimas en términos de número de pasos. Adicionalmente, se incluyó el uso de un repositorio en GitHub para gestionar versiones de los códigos, documentar los avances y evidenciar la mejora continua a lo largo del desarrollo del laboratorio. Con este enfoque, se consolidó un flujo de trabajo que refleja tanto la evolución técnica de los algoritmos como el proceso de aprendizaje académico.

II. Objetivos

II-A. Objetivo General

Implementar y analizar problemas clásicos de Inteligencia Artificial mediante la representación en espacios de estados y la aplicación de algoritmos de búsqueda, con el fin de comprender los fundamentos de la resolución automática de problemas.

II-B. Objetivos Específicos

- Representar el problema del *8-puzzle* en un espacio de estados y resolverlo utilizando el algoritmo de Búsqueda en Anchura (BFS), verificando la validez de los estados y la secuencia de movimientos hacia la solución.
- Modelar ejemplos sencillos de espacios de estados y acciones (lámpara, mascota virtual y búsqueda del tesoro) para afianzar los conceptos de estado inicial, acciones disponibles y estado meta, aplicando BFS cuando es necesario.
- Desarrollar la representación de un laberinto en mundos discretos de 2×2 y 3×3, considerando obstáculos, tablas de transición y recompensas asociadas, y aplicar BFS para encontrar rutas óptimas y analizar métricas de desempeño.

III. Procedimiento

El desarrollo del laboratorio se estructuró en tres apartados, cada uno con un nivel creciente de complejidad, pero todos fundamentados en la noción de espacio de estados, acciones disponibles y estado meta. La estrategia general consistió en representar formalmente cada problema como un grafo implícito y aplicar algoritmos de búsqueda adecuados para garantizar la obtención de soluciones óptimas en número de pasos.

En el primer apartado, correspondiente al *8-puzzle*, se definieron los estados como configuraciones posibles de un tablero de 3×3 con ocho fichas numeradas y un espacio vacío. Las acciones se establecieron como movimientos válidos del espacio vacío hacia arriba, abajo, izquierda o derecha. A partir de estas definiciones, el problema se abordó como un grafo de estados, aplicando el algoritmo de Búsqueda en Anchura (BFS) para recorrer sistemáticamente los estados hasta encontrar el estado meta. Adicionalmente, se consideró la verificación de la resolubilidad del estado inicial frente al estado objetivo, lo cual garantiza que los recursos computacionales no se inviertan en problemas intrínsecamente imposibles.

En el segundo apartado se trabajaron ejemplos de menor complejidad conceptual con el objetivo de afianzar los fundamentos de los espacios de estados. Se definió un modelo de dos estados para el caso de la lámpara (encendida/apagada), un modelo para la mascota virtual (contenta/triste) y un modelo de cuadrícula para la búsqueda de un tesoro. En cada caso se identificaron explícitamente los estados posibles, las acciones de transición y el estado meta. Estos ejemplos permitieron reforzar la comprensión del ciclo básico de percepción-acción en un agente y sirvieron como preparación para problemas más complejos.

En el tercer apartado se abordó la representación de un laberinto, inicialmente en un mundo 2×2 y posteriormente en un mundo 3×3. En este contexto, los estados fueron representados como coordenadas en la cuadrícula, las acciones como movimientos de desplazamiento, y los estados meta como posiciones objetivo dentro del entorno. Se introdujo la noción de obstáculos que restringen el espacio de estados alcanzable y se construyó una tabla de transiciones para representar explícitamente las posibilidades de movimiento desde cada estado. Finalmente, se incluyó un esquema de recompensas asociado a las transiciones, de modo que cada paso implicara un costo y alcanzar la meta proporcionara una ganancia.

Adicionalmente, todos los códigos desarrollados fueron gestionados mediante un repositorio en GitHub, lo que permitió llevar un control detallado de las versiones, registrar los avances y documentar las mejoras continuas realizadas en cada commit. De esta manera, se consolidó un flujo de trabajo que refleja tanto la evolución de los algoritmos como el proceso de aprendizaje asociado a su construcción.

En todos los apartados, la estrategia de solución se basó en el algoritmo de Búsqueda en Anchura, dado que este garantiza encontrar la ruta más corta desde el estado inicial hasta el estado meta. De este modo, el laboratorio permitió aplicar de manera práctica conceptos teóricos de representación de problemas y estrategias de búsqueda en Inteligencia Artificial.

IV. Resultados

IV-A. Punto 1: 8-Puzzle (BFS)

La ejecución del programa correspondiente al 8-puzzle arrojó la siguiente salida, donde se observa el número de pasos óptimos, la secuencia de movimientos y la traza de tableros visitados en la solución:

```
PS C:\Users\PC\OneDrive\Documents\Universidad Santo Tomas\Semestre 8 - 2025 1\Intro
    IA> & C:/Users/PC/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/PC/
    OneDrive/Documents/Universidad Santo Tomas/Semestre 8 - 2025 1/Intro IA/Corte 1/
    Laboratorio_1/Primer_punto.py"
Pasos (Optimos): 5
Movimientos: arriba -> arriba -> izquierda -> abajo -> derecha
Nodos expandidos (BFS): 58

Secuencia de tableros:
2 8 3
1 6 4
7 0 5

2 8 3
1 0 4
7 6 5

2 0 3
1 8 4
7 6 5

0 2 3
1 8 4
7 6 5

1 2 3
0 8 4
7 6 5

1 2 3
8 0 4
7 6 5
```

Listing 1: Salida de ejecución – Punto 1 (8-Puzzle)

IV-B. Punto 2: Espacio de estados y acciones (Lámpara, Mascota, Tesoro)

La siguiente salida evidencia el cumplimiento de metas en los ejemplos simples (lámpara y mascota), así como el hallazgo de la ruta mínima en la cuadrícula para el caso del tesoro con BFS y sus métricas:

```
PS C:\Users\PC\OneDrive\Documents\Universidad Santo Tomas\Semestre 8 - 2025 1\Intro
    IA> & C:/Users/PC/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/PC/
    OneDrive/Documents/Universidad Santo Tomas/Semestre 8 - 2025 1/Intro IA/Corte 1/
    Laboratorio_1/Segundo_punto.py"
=== LAMPARA (espacio de estados y acciones) ===
Estados: ['ENCENDIDA', 'APAGADA']
Inicial: APAGADA | Meta: ENCENDIDA
Acciones disponibles: ['PRENDER', 'APAGAR']
Aplicar PRENDER -> ENCENDIDA
Meta alcanzada!

=== MASCOTA (recompensa & ambiente) ===
```

```
Estados: ['CONTENTA', 'TRISTE']
Inicial: TRISTE | Meta: CONTENTA
AcciOn: DAR_COMIDA -> Estado: CONTENTA | Recompensa: 1
Meta alcanzada!

=== TESORO (BFS con mEtricas) ===
Inicio: (0, 0)  |  Meta: (2, 2)
Acciones: -> -> ABAJO ABAJO
Camino: (0, 0) -> (0, 1) -> (0, 2) -> (1, 2) -> (2, 2)
Metricas: 9 nodos expandidos | longitud de camino = 4

Tesoro encontrado!
```

Listing 2: Salida de ejecución – Punto 2 (Lámpara, Mascota, Tesoro)

IV-C. Punto 3: Laberinto 3x3 con obstáculos, tabla de transición y recompensas

Para el laberinto 3×3 con un obstáculo en el centro, se obtuvo la ruta óptima, junto con métricas de desempeño y un ejemplo de la tabla de recompensas:

```
PS C:\Users\PC\OneDrive\Documents\Universidad Santo Tomas\Semestre 8 - 2025 1\Intro
   IA> & C:/Users/PC/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/PC/
   OneDrive/Documents/Universidad Santo Tomas/Semestre 8 - 2025 1/Intro IA/Corte 1/
   Laboratorio_1/Segundo_punto.py"
=== LABERINTO 3x3 - version final ===
Tamano: 3x3 | Inicio: (0, 0) | Meta: (2, 2) | Obstaculos: {(1, 1)}
Acciones: -> -> ABAJO ABAJO
Camino : (0, 0) -> (0, 1) -> (0, 2) -> (1, 2) -> (2, 2)
Metricas: 8 nodos expandidos | longitud camino = 4
Ejemplo de recompensas (primeros 5): [(((0, 0), '->', (0, 1)), -1), (((0, 0), 'abajo
   ', (1, 0)), -1), (((0, 1), '->', (0, 2)), -1), (((0, 1), '<-', (0, 0)), -1), (((0,
    2), 'abajo', (1, 2)), -1)]
```

Listing 3: Salida de ejecución – Punto 3 (Laberinto 3x3 con obstáculos)

V. Análisis de resultados

V-A. Punto 1: 8-puzzle con Búsqueda en Anchura (BFS)

V-A1. Modelado del problema: El estado se modeló como una matriz 3×3 de enteros en la que el valor 0 representa el hueco. Se fijó explícitamente el estado objetivo (*goal*) de acuerdo con el enunciado, lo que asegura que todas las decisiones de búsqueda y verificación de resolubilidad se orienten al mismo arreglo final.

```
N = 3
GOAL = [
    [1, 2, 3],
    [8, 0, 4],
    [7, 6, 5]
]

ROW = [0, 0, -1, 1]
COL = [-1, 1, 0, 0]
MOVE_NAME = ["izquierda", "derecha", "arriba", "abajo"]
```

Listing 4: Constantes, objetivo y mapeo de movimientos

Las listas `ROW` y `COL` codifican desplazamientos relativos del hueco en las cuatro direcciones. La separación de los nombres de movimiento (`MOVE_NAME`) facilita reconstruir una ruta *legible* para el informe. El estado se encapsula en una clase minimalista para transportar información operativa (tablero, posición del hueco y profundidad).

```
class PuzzleState:
    def __init__(self, board, x, y, depth):
        self.board = board
        self.x = x
        self.y = y
        self.depth = depth
```

Listing 5: Estructura mínima del estado

V-A2. Chequeo de resolubilidad: Antes de ejecutar BFS se verifica si el estado inicial es resoluble hacia el objetivo dado. Para 8-puzzle en tablero impar (3x3), la condición es que la *paridad de inversiones* del inicial, medida *relativamente al orden del objetivo*, sea par. Así se evita explorar un espacio imposible.

```
def is_solvable(initial, goal):
    """
    8-puzzle 3x3: el estado inicial es resoluble hacia 'goal' si la paridad
    de inversiones del inicial relativa al orden del goal es PAR.
    """
    def flat_wo_zero(m):
        return [x for r in m for x in r if x != 0]

    goal_order = {v: i for i, v in enumerate(flat_wo_zero(goal))}
    seq = [goal_order[v] for v in flat_wo_zero(initial)]
    inv = sum(1 for i in range(len(seq)) for j in range(i+1, len(seq)) if seq[i] >
   seq[j])
    return inv % 2 == 0
```

Listing 6: Verificación de resolubilidad

Justificación: en 3x3 el hueco no altera la paridad; reindexar por el orden del *goal* canoniza la comparación. Si el número de inversiones es impar, no existe ninguna secuencia de movimientos que lleve del estado inicial al objetivo.

V-A3. Diseño del BFS y estructuras auxiliares: El algoritmo usa una cola FIFO (`deque`) para explorar por niveles; un conjunto `visited` de estados *hashables* (tupla de tuplas) para evitar ciclos/repeticiones; y un diccionario `parents` para reconstruir la ruta óptima al alcanzar la meta. También se contabilizan métricas de desempeño.

```python
from collections import deque

def solve_puzzle_bfs(start):
    if not is_solvable(start, GOAL):
        return None, None, {"expandidos": 0, "profundidad": None}

    zx, zy = next((i, j) for i in range(N) for j in range(N) if start[i][j] == 0)

    start_key = tuple(map(tuple, start))
    goal_key  = tuple(map(tuple, GOAL))

    q = deque([PuzzleState(start, zx, zy, 0)])
    visited = {start_key}
    parents = {start_key: (None, None)}  # hijo -> (padre, movimiento)
    expanded = 0
```

Listing 7: Inicialización de BFS, visitados, padres y métricas

Justificación de diseño:

- *Representación hashable*: convertir la matriz a tuplas permite guardar estados en `set`/`dict` con costo amortizado $O(1)$ en búsqueda/inserción.
- *Estructura* `parents`: habilita reconstrucción de camino sin almacenar rutas completas en cada nodo (ahorro de memoria).
- *Métrica* `expanded`: reporta cuántos nodos fueron realmente *extraídos* de la cola y expandidos, útil para comparar estrategias.

V-A4. Expansión de sucesores: Cada expansión mueve el hueco en hasta cuatro direcciones válidas; se genera un nuevo tablero por copia superficial por fila (para no mutar estados ancestros), se etiqueta el movimiento y se encola si aún no ha sido visitado.

```python
while q:
        curr = q.popleft()
        expanded += 1
        curr_key = tuple(map(tuple, curr.board))

        if curr_key == goal_key:
            # reconstruccion
            path_states, path_moves = [], []
            k = curr_key
            while k is not None:
                parent, move = parents[k]
                path_states.append([list(r) for r in k])
                path_moves.append(move)
                k = parent
            path_states.reverse()
            path_moves = [m for m in reversed(path_moves)][1:]

            stats = {"expandidos": expanded, "profundidad": len(path_moves)}
            return path_states, path_moves, stats

        for i in range(4):
            nx, ny = curr.x + ROW[i], curr.y + COL[i]
            if is_valid(nx, ny):
                new_board = [r[:] for r in curr.board]
                new_board[curr.x][curr.y], new_board[nx][ny] = new_board[nx][ny],
    new_board[curr.x][curr.y]
                key = tuple(map(tuple, new_board))
                if key not in visited:
```

```
                    visited.add(key)
                    parents[key] = (curr_key, MOVE_NAME[i])
                    q.append(PuzzleState(new_board, nx, ny, curr.depth + 1))

    return None, None, {"expandidos": expanded, "profundidad": None}
```

Listing 8: Bucle principal de expansión y generación de vecinos

Decisiones clave:

1. *Copia por filas* (`[r[:] for r in curr.board]`): evita aliasing entre estados y garantiza inmutabilidad *lógica* del padre.
2. *Marcado temprano en* `visited` (al generar, no al *expandir*): previene encolados duplicados del mismo estado en el mismo nivel, reduciendo memoria y tiempo.
3. *Coste uniforme*: cada acción cuesta 1, por lo que BFS garantiza *óptimo en número de pasos*.

V-A5. Criterio de parada y reconstrucción de la ruta: Al alcanzar la clave del objetivo, se retrocede usando `parents` hasta el estado inicial y se invierte la secuencia.

```
if curr_key == goal_key:
    path_states, path_moves = [], []
    k = curr_key
    while k is not None:
        parent, move = parents[k]
        path_states.append([list(r) for r in k])
        path_moves.append(move)
        k = parent
    path_states.reverse()
    path_moves = [m for m in reversed(path_moves)][1:]

    stats = {"expandidos": expanded, "profundidad": len(path_moves)}
    return path_states, path_moves, stats
```

Listing 9: Reconstrucción de la solución y estadísticos

Observación: la docstring de `solve_puzzle_bfs` indicaba inicialmente que retornaba dos valores; en esta versión final retorna *tres* (tableros, movimientos, métricas). Es recomendable actualizar la documentación interna para mantener consistencia.

V-A6. Correctitud y optimalidad: Dado que:

1. el grafo de estados está no ponderado (costo uniforme por transición),
2. `visited` evita revisitar estados con costo mayor o igual,
3. y BFS explora por capas (niveles de profundidad),

entonces el primer encuentro del objetivo ocurre a profundidad mínima, y la ruta reconstruida tiene el número mínimo de movimientos. El chequeo de resolubilidad garantiza que no haya falsos negativos debidos a paridades incompatibles entre inicio y meta.

V-A7. Resultados obtenidos y lectura: En la ejecución suministrada para el estado inicial $\begin{bmatrix} 2 & 8 & 3 \\ 1 & 6 & 4 \\ 7 & 0 & 5 \end{bmatrix}$, el algoritmo reportó:

- **Pasos (óptimos)**: 5.
- **Movimientos**: `arriba` → `arriba` → `izquierda` → `abajo` → `derecha`.
- **Nodos expandidos**: 58.

La secuencia de tableros muestra una progresión coherente con estos movimientos y valida que el *goal* elegido (List. 4) fue alcanzado. El número de nodos expandidos es moderado en este caso porque el estado inicial está relativamente *cerca* del objetivo (profundidad 5). No obstante, en instancias más alejadas el tamaño de frontera puede crecer rápidamente.

V-A8. Complejidad y limitaciones: Bajo costo uniforme, la complejidad temporal de BFS es $O(b^d)$ y la espacial también es $O(b^d)$, donde b es el factor de ramificación y d la profundidad de la solución. En 8-puzzle, el espacio de estados resolubles es $\frac{9!}{2} = 181{,}440$ estados; aunque BFS es correcto y óptimo, su consumo de memoria puede ser significativo para profundidades altas. En contextos más exigentes, estrategias informadas como A* con heurísticas admisibles (p. ej., *Manhattan* o *mal colocadas*) reducen drásticamente el número de expansiones, manteniendo optimalidad.

V-A9. Decisiones de ingeniería:

- **Separación de responsabilidades**: funciones específicas para validación, impresión, solvencia y búsqueda favorecen claridad y mantenibilidad.
- **Estructuras eficientes**: `deque`, `set` y `dict` ofrecen operaciones amortizadas $O(1)$ críticas para rendimiento.
- **Métricas integradas**: `expandidos` y `profundidad` facilitan el análisis comparativo entre variantes y futuros trabajos.

En suma, la solución implementada cumple los objetivos propuestos: representa correctamente el problema, verifica la factibilidad, encuentra una ruta óptima con BFS y reporta métricas útiles para el análisis académico.

V-B. Punto 2: Espacios de estados y acciones simples

Este segundo punto tuvo como objetivo reforzar el concepto de espacios de estados y acciones a través de tres ejemplos sencillos: la lámpara, la mascota virtual y la búsqueda de un tesoro en una cuadrícula. Aunque conceptualmente menos complejos que el 8-puzzle, estos casos permiten asentar los fundamentos de modelado de problemas como grafos de estados, con transiciones definidas y estados meta.

V-B1. Caso 1: Lámpara: La lámpara se modela como un sistema con dos estados posibles: `ENCENDIDA` y `APAGADA`. Las acciones posibles son `PRENDER` y `APAGAR`. La función de transición se define de forma determinista.

```python
def acciones_lampara(_estado):
    return ["PRENDER", "APAGAR"]

def transicion_lampara(estado, accion):
    if accion == "PRENDER":
        return "ENCENDIDA"
    if accion == "APAGAR":
        return "APAGADA"
    return estado
```

Listing 10: Acciones y transición de la lámpara

Análisis:

- La lámpara es un ejemplo minimalista de espacio de estados.
- La transición es determinista y sin restricciones: siempre se puede pasar de un estado al otro.
- Justificación: este caso ilustra la noción de estado inicial, acción y estado meta en su forma más simple.

V-B2. Caso 2: Mascota virtual: La mascota es un modelo igualmente binario con estados `CONTENTA` y `TRISTE`. La acción `DAR_COMIDA` conduce a un estado positivo, mientras que `QUITAR_COMIDA` produce el contrario. Se incorpora además una noción de *recompensa*, que asigna +1 al alcanzar `CONTENTA`.

```python
def transicion_mascota(estado, accion):
    if accion == "DAR_COMIDA":
        return "CONTENTA"
    if accion == "QUITAR_COMIDA":
        return "TRISTE"
    return estado

def recompensa_mascota(estado, accion, nuevo_estado):
    return 1 if nuevo_estado == "CONTENTA" else 0
```

Listing 11: Acciones, transición y recompensa de la mascota

Análisis:

- Se introduce el concepto de **recompensa**, fundamental en aprendizaje por refuerzo.
- Aunque el modelo es simple, ilustra la asociación entre transición de estados y valor de la acción.
- Justificación: con este caso se prepara al estudiante para entender más adelante cómo agentes aprenden políticas en base a recompensas.

V-B3. Caso 3: Tesoro en cuadrícula 3x3: Este ejemplo expande el modelo a un entorno espacial: un agente debe ir desde $(0, 0)$ hasta $(2, 2)$ en una cuadrícula de 3×3. Se define el conjunto de movimientos posibles (arriba, abajo, izquierda, derecha) y se aplica BFS para hallar el camino más corto.

```python
MOVES = [(-1,0,"arriba"), (0,1,"->"), (1,0,"abajo"), (0,-1,"<-")]

def bfs_camino(inicio, meta):
    q = deque([inicio])
    parents = {inicio: (None, None)}  # (padre, accion)
    visit = {inicio}
    expandidos = 0
```

```
8
9       while q:
10          x, y = q.popleft()
11          expandidos += 1
12
13          if (x, y) == meta:
14              # reconstruccion del camino
15              ...
```

Listing 12: Movimientos y BFS en cuadrícula

Análisis:

- El uso de BFS garantiza la ruta más corta porque cada movimiento tiene costo uniforme.
- La estructura `parents` permite reconstruir tanto la secuencia de posiciones como de acciones.
- El diccionario `stats` añade métricas (nodos expandidos, longitud de camino) útiles para evaluar eficiencia.
- Justificación: este caso generaliza la idea de estados y acciones a un entorno espacial con múltiples caminos, mostrando cómo algoritmos de búsqueda sistemáticos encuentran soluciones óptimas.

V-B4. Discusión global del Punto 2:

- La lámpara enseña **transiciones deterministas simples**.
- La mascota introduce **transiciones con recompensas**.
- El tesoro incorpora **espacios más grandes**, **algoritmos de búsqueda** y **métricas**.

En conjunto, los tres ejemplos muestran un gradiente de complejidad creciente que ilustra cómo cualquier problema de IA puede reducirse a: definir estados, acciones y metas, y aplicar un método de búsqueda para alcanzar la solución.

V-C. Punto 3: Laberinto en mundos 2x2 y 3x3

El tercer punto del laboratorio tuvo como objetivo extender la noción de espacios de estados a entornos espaciales más realistas, representados como cuadrículas. Se inició con un mundo 2×2 y posteriormente se generalizó a un mundo 3×3. La versión final incluye obstáculos, tablas de transición, recompensas y métricas de desempeño.

V-C1. Configuración del mundo y acciones: El mundo se define como una cuadrícula de tamaño 3×3. Cada estado corresponde a una celda con coordenadas (x, y). Las acciones posibles son movimientos cardinales, representados en un diccionario que asocia el símbolo con el desplazamiento relativo.

```
W, H = 3, 3
ACCIONES = {
    "arriba": (-1, 0),
    "->": (0, +1),
    "abajo": (+1, 0),
    "<-": (0, -1),
}
OBSTACULOS = {(1, 1)}  # bloqueamos el centro
INICIO = (0, 0)
META   = (2, 2)
```

Listing 13: Definición del mundo y acciones

Análisis:

- `ACCIONES` proporciona un mapa explícito entre acción simbólica y su efecto sobre las coordenadas.
- `OBSTACULOS` permite restringir el espacio de estados alcanzable, mostrando cómo los obstáculos modifican la dinámica del agente.
- `INICIO` y `META` fijan el problema de búsqueda.

V-C2. Construcción de la tabla de transiciones: Para cada celda del grid se construye un diccionario de posibles transiciones. Si una celda es obstáculo, se omiten sus sucesores. Si una acción lleva fuera de la cuadrícula o a un obstáculo, se asigna `None`.

```
def construir_transiciones(obstaculos):
    T = {}
    for x in range(W):
        for y in range(H):
            s = (x, y)
            T[s] = {}
            if s in obstaculos:
                continue
            for a, (dx, dy) in ACCIONES.items():
                nx, ny = x + dx, y + dy
                T[s][a] = (nx, ny) if dentro(nx, ny) and (nx, ny) not in obstaculos
   else None
    return T
```

Listing 14: Construcción de la tabla de transiciones

Justificación: construir una tabla de transiciones explicita el grafo del problema y facilita la visualización de los estados alcanzables. Además, permite depurar fácilmente la dinámica antes de ejecutar búsquedas.

V-C3. Recompensas asociadas a transiciones: Se definió un esquema de recompensas donde cada paso cuesta -1 y alcanzar la meta otorga $+10$. El mapeo se implementa como un diccionario cuyas llaves son tuplas $(estado, accin, nuevo\_estado)$.

```
def recompensas(meta):
    R = {}
    for x in range(W):
        for y in range(H):
            s = (x, y)
            for a, (dx, dy) in ACCIONES.items():
```

```
                nx, ny = x + dx, y + dy
                if not dentro(nx, ny) or (nx, ny) in OBSTACULOS:
                    continue
                ns = (nx, ny)
                R[(s, a, ns)] = 10 if ns == meta else -1
    return R
```

Listing 15: Función de recompensas

Análisis:

- Este esquema permite cuantificar el costo de alcanzar la meta.
- Aunque el laboratorio no implementa un algoritmo de aprendizaje, la presencia de recompensas muestra la transición hacia técnicas de aprendizaje por refuerzo.

V-C4. Búsqueda con BFS: Se utilizó nuevamente BFS por su simplicidad y garantía de encontrar la ruta mínima en número de pasos. Se mantiene la misma estructura conceptual de padres y visitados vista en el 8-puzzle.

```
def bfs(inicio, meta, T):
    q = deque([inicio])
    padres = {inicio: (None, None)}
    visit  = {inicio}
    expandidos = 0

    while q:
        u = q.popleft()
        expandidos += 1
        if u == meta:
            # reconstruccion
            path, acts = [], []
            cur = u
            while cur is not None:
                p, a = padres[cur]
                path.append(cur); acts.append(a)
                cur = p
            path.reverse()
            acts = [a for a in reversed(acts)][1:]
            return path, acts, {"expandidos": expandidos, "long_camino": len(acts)}

        for v, a in vecinos(u, T):
            if v not in visit:
                visit.add(v)
                padres[v] = (u, a)
                q.append(v)
```

Listing 16: Algoritmo BFS con reconstrucción y métricas

Justificación:

- BFS explora primero todos los caminos de menor longitud, garantizando una solución óptima en costo uniforme.
- La métrica `expandidos` permite cuantificar el esfuerzo de búsqueda; `long_camino` reporta la calidad de la solución.

V-C5. Resultados y discusión: En la ejecución con obstáculo en el centro $(1,1)$, el agente encontró una ruta alternativa desde $(0,0)$ hasta $(2,2)$ en 4 pasos:

$$(0,0) \to (0,1) \to (0,2) \to (1,2) \to (2,2)$$

- **Acciones**: $\to\to\downarrow\downarrow$.
- **Nodos expandidos**: 8.
- **Longitud de camino**: 4.

- **Recompensas**: los primeros pares reportados evidencian el costo negativo de cada paso y la ganancia final en la meta.

V-C6. Conclusiones parciales del Punto 3:

- La representación con tabla de transiciones facilita visualizar dinámicas de movimiento y restricciones.
- La incorporación de recompensas conecta los problemas de búsqueda con los de decisión y aprendizaje.
- BFS resultó suficiente para entornos pequeños, aunque en entornos mayores sería necesario evaluar algoritmos más eficientes como A*.

VI. Conclusiones

El desarrollo del laboratorio permitió cumplir de manera satisfactoria el objetivo general y los objetivos específicos planteados, consolidando la comprensión del concepto de espacio de estados, la formulación de acciones y metas, así como la aplicación de algoritmos de búsqueda.

- Se logró representar y resolver el *8-puzzle* utilizando Búsqueda en Anchura (BFS). La verificación de resolubilidad, la reconstrucción de la ruta óptima y la inclusión de métricas (nodos expandidos y profundidad) evidenciaron la validez del enfoque y reforzaron la importancia del control de estados visitados en la búsqueda.
- Los ejemplos sencillos de la lámpara, la mascota virtual y la búsqueda del tesoro permitieron afianzar los conceptos de estado, acción y meta. Estos casos demostraron que incluso sistemas de baja complejidad son útiles para comprender principios fundamentales de Inteligencia Artificial. En particular, la introducción de recompensas en el caso de la mascota mostró cómo conectar la transición de estados con la noción de valor.
- El problema del laberinto en mundos de 2×2 y 3×3 ilustró cómo los obstáculos y las tablas de transición afectan la dinámica de búsqueda. El uso de recompensas permitió extender el análisis hacia contextos propios del aprendizaje por refuerzo. BFS garantizó la obtención de soluciones óptimas, aunque se reconoce que para entornos más complejos podrían ser necesarios algoritmos informados como A*.
- La gestión del proyecto en un repositorio de GitHub facilitó el control de versiones, el registro de avances y la mejora continua de los códigos. Este aspecto no solo fortaleció la organización del trabajo, sino que además constituyó una práctica alineada con estándares de desarrollo profesional.

En síntesis, el laboratorio cumplió con los objetivos propuestos al demostrar que la correcta representación de un problema como espacio de estados y la selección de un algoritmo de búsqueda adecuado son elementos centrales en la resolución automática de problemas en Inteligencia Artificial.

Referencias

[1] D. Bergmann, "State space models," IBM, 07-jul-2025. [En línea]. Disponible en: https://www.ibm.com/think/topics/state-space-model. [Consultado: 15-ago-2025].

[2] GeeksforGeeks, "8 Puzzle Problem using Branch and Bound," [En línea]. Disponible en: https://www.geeksforgeeks.org/dsa/8-puzzle-problem-using-branch-and-bound/. [Consultado: 15-ago-2025].

[3] J. A. Alonso Jiménez, "Tema 4: Resolución de problemas de espacios de estados," Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad de Sevilla, Curso 2004–05. [En línea]. Disponible en: http://www.cs.us.es/~jalonso. [Consultado: 15-ago-2025].

[4] S. Russell y P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Pearson, 2010.

[5] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006. [En línea]. Disponible en: http://planning.cs.uiuc.edu/. [Consultado: 15-ago-2025].

[6] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997.