PRÁCTICA DE JUEGO UNIPERSONAL

(Inteligencia Artificial para Juegos)

Juego 2: Water Sort Puzzle - Resolución mediante búsqueda en espacio de estados

Introducción

En esta práctica, se les pide implementar el juego **Water Sort Puzzle** utilizando conceptos de búsqueda en espacios de estados y heurísticas para su resolución automática. El objetivo es aplicar los conceptos aprendidos en el curso de Inteligencia Artificial para Juegos, como la representación de problemas mediante espacios de estados, implementación de algoritmos de búsqueda (amplitud, profundidad, A*, IDA*) y el diseño de funciones heurísticas efectivas. Puede verse una versión funcional del juego en https://watersort.io/es/

Descripción del juego Water Sort Puzzle

El Water Sort Puzzle es un rompecabezas lógico donde el jugador debe organizar líquidos de diferentes colores en tubos de ensayo. El juego sigue estas reglas:

Configuración inicial:

- Se tienen **N tubos de ensayo** (típicamente entre 5 y 12)
- Cada tubo tiene capacidad para **4 unidades de líquido**
- Inicialmente hay **K colores diferentes** distribuidos aleatoriamente
- Algunos tubos comienzan vacíos (generalmente 2)
- Los líquidos se apilan verticalmente en los tubos

Reglas del juego:

1. **Movimiento válido:**
    - Solo se puede verter líquido desde la parte superior de un tubo origen a otro tubo destino
    - El tubo destino debe tener espacio disponible
    - Solo se puede verter si:
        - El tubo destino está vacío, o
        - El color superior del tubo destino coincide con el color que se va a verter
2. **Vertido continuo:**
    - Al realizar un movimiento, se vierte TODO el líquido continuo del mismo color desde la parte superior del tubo origen
    - Ejemplo: Si un tubo tiene [Azul, Azul, Rojo, Verde] desde arriba, se vierten las 2 unidades de Azul
3. **Condición de victoria:**
    - El juego se resuelve cuando cada tubo está completamente vacío o contiene exactamente 4 unidades del mismo color
4. **Condición de derrota:**
    - No hay movimientos válidos disponibles y no se ha alcanzado la condición de victoria

Objetivos de la práctica

1. Modelado del espacio de estados:

- **Estado:** Representar cada tubo como una lista/pila de colores (máximo 4 elementos)
- **Estado inicial:** Configuración aleatoria pero resoluble con semilla
- **Estado objetivo:** Todos los tubos vacíos o llenos con un solo color
- **Operadores:** Todos los movimientos válidos de verter de un tubo a otro

2. Algoritmos de búsqueda a implementar:

Obligatorios:

- Búsqueda en amplitud (BFS)
- Búsqueda en profundidad (DFS)
- Búsqueda con heurística A*

Opcionales:

- Búsqueda con profundidad limitada
- IDA* (Iterative Deepening A*)
- Backtracking con cota

3. Funciones heurísticas:

Diseñar al menos **dos heurísticas diferentes** y comparar su rendimiento:

Heurística 1 - Entropía de colores:

- Penalizar la dispersión de colores iguales en diferentes tubos
- Favorecer estados donde los colores estén más agrupados

Fórmula:

$$h_1(\text{estado}) = \Sigma_i \ (\text{cantidad\_tubos\_con\_color}_i - 1) \times \text{peso\_dispersión}_i$$

donde:
- $\text{cantidad\_tubos\_con\_color}_i$ = número de tubos que contienen el color i
- peso_dispersión_i = cantidad de unidades del color i que no están en su tubo "principal"
- tubo principal = tubo con mayor cantidad del color i

Ejemplo de cálculo:

```
Estado:
Tubo 0: [R, R, A, V]
Tubo 1: [A, V, V, R]
Tubo 2: [A, R, A, V]
Tubo 3: []
Tubo 4: []

Análisis por color:
```

```
- Rojo (R): aparece en 3 tubos (0,1,2)
  * Tubo principal: Tubo 0 (2 rojos)
  * Unidades fuera del principal: 2 (1 en tubo 1, 1 en tubo 2)
  * Contribución: (3-1) × 2 = 4

- Azul (A): aparece en 3 tubos (0,1,2)
  * Tubo principal: Tubo 2 (2 azules)
  * Unidades fuera del principal: 2 (1 en tubo 0, 1 en tubo 1)
  * Contribución: (3-1) × 2 = 4

- Verde (V): aparece en 3 tubos (0,1,2)
  * Tubo principal: Tubo 1 (2 verdes)
  * Unidades fuera del principal: 2 (1 en tubo 0, 1 en tubo 2)
  * Contribución: (3-1) × 2 = 4

h₁(estado) = 4 + 4 + 4 = 12
```

Heurística 2 - Tubos completados:

- Valorar positivamente el número de tubos completados (vacíos o con 4 del mismo color)
- Considerar el número de movimientos necesarios estimados para completar tubos parciales

Fórmula:

```
h₂(estado) = (tubos_incompletos × 4) - colores_bien_posicionados
```

donde:
- tubos_incompletos = número de tubos que NO están vacíos ni tienen 4 del mismo color
- colores_bien_posicionados = suma de colores consecutivos desde el fondo en cada tubo

Ejemplo de cálculo:

```
Estado:
Tubo 0: [R, R, A, V]  → Incompleto, 1 bien posicionado (V en el fondo)
Tubo 1: [A, V, V, R]  → Incompleto, 1 bien posicionado (R en el fondo)
Tubo 2: [A, R, A, V]  → Incompleto, 1 bien posicionado (V en el fondo)
Tubo 3: []            → Completo (vacío)
Tubo 4: []            → Completo (vacío)

Tubos incompletos = 3 (tubos 0, 1, 2)
Colores bien posicionados = 1 + 1 + 1 = 3

h₂(estado) = (3 × 4) - 3 = 12 - 3 = 9
```

Heurística 3 (opcional) - Bloqueos:

- Detectar y penalizar configuraciones que generan bloqueos
- Valorar la "libertad de movimiento" disponible

Fórmula:

```
h₃(estado) = total_unidades_mezcladas + (2 × unidades_bloqueadas)

donde:
- total_unidades_mezcladas = número de unidades en tubos que tienen más de un
color
```

```
- unidades_bloqueadas = unidades que tienen encima al menos una unidad de
diferente color

Explicación:
- Un tubo "puro" (vacío o con un solo color) tiene 0 unidades mezcladas
- Un tubo con múltiples colores tiene todas sus unidades contando como
mezcladas
- Una unidad está bloqueada si no puede moverse directamente (tiene otros
colores encima)
```

Ejemplo de cálculo:

```
Estado:
Tubo 0: [R, R, A, V]  → 4 unidades mezcladas (tubo con múltiples colores)
                        Bloqueadas: V bloqueado por ARR (3 unidades encima)
                                    A bloqueado por RR (2 unidades encima)
                                    R no bloqueados (0 unidad encima)
                                    Total: 3 + 2  = 5 unidades bloqueando
Tubo 1: [A, V, V, R]  → 4 unidades mezcladas (tubo con múltiples colores)
                        Bloqueadas: R bloqueado por AVV (3 unidades encima)
                                    1er V bloqueado por A (1 unidad encima)
                                    2do V bloqueado por A (1 unidad encima)
                                    Total: 3 + 1 + 1 = 5 unidades bloqueando
Tubo 2: [A, R, A, V]  → 4 unidades mezcladas (tubo con múltiples colores)
                        Bloqueadas: V bloqueado por ARA (3 unidades encima)
                                    A(fondo) bloqueado por R (1 unidad encima)
                                    R bloqueado por A (1 unidad encima)
                                    Total: 3 + 1 + 1 = 5 unidades bloqueando
Tubo 3: []            → 0 unidades mezcladas (vacío = puro)
Tubo 4: []            → 0 unidades mezcladas (vacío = puro)

Total unidades mezcladas = 4 + 4 + 4 + 0 + 0 = 12
Total unidades bloqueando = 5 + 5 + 5 = 15

h₃(estado) = 12 + (2 × 15) = 12 + 30 = 42
```

4. Análisis y evaluación:

- Comparar eficiencia de los algoritmos (nodos expandidos, tiempo, memoria)
- Evaluar calidad de las soluciones (número de movimientos)
- Analizar el impacto de diferentes heurísticas

Especificaciones de implementación

1. Estructura del programa:

```
class WaterSortGame:
    def __init__(self, num_tubes, num_colors, seed=None)
    def get_valid_moves(self, state)
    def apply_move(self, state, move)
    def is_goal_state(self, state)
    def is_valid_state(self, state)

class SearchAlgorithm:
    def bfs(self, initial_state)
    def dfs(self, initial_state)
```

```
def a_star(self, initial_state, heuristic)
def ida_star(self, initial_state, heuristic)  # Opcional
```

2. Input/Output:

Input:

- Número de tubos (entre 5 y 12)
- Número de colores (entre 3 y num_tubos-2)
- Semilla para reproducibilidad
- Selección de algoritmo y heurística
- Modo de ejecución (paso a paso o automático)

Output:

- Visualización del estado inicial
- Secuencia de movimientos solución
- Estadísticas de búsqueda:
  - Nodos expandidos
  - Nodos en memoria máxima
  - Tiempo de ejecución
  - Profundidad de la solución
- Visualización del proceso (opcional con pygame o ASCII art)

3. Requisitos técnicos:

- Python 3.9+
- Uso de estructuras de datos eficientes (heapq para A*, deque para BFS)
- Documentación clara del código
- Manejo de estados mediante tuplas inmutables o hashing apropiado
- Detección de estados repetidos para evitar ciclos

Criterios de evaluación

Criterio	Descripción
Corrección	El programa resuelve correctamente el puzzle siguiendo las reglas
Algoritmos	Implementación correcta de BFS, DFS y A* como mínimo
Heurísticas	Diseño e implementación de al menos 2 heurísticas efectivas
Eficiencia	Optimización del código y uso eficiente de memoria
Documentación	Código bien documentado, informe completo con análisis

Entregar:

1. **Código fuente:**
   - Archivo principal: `water_sort_solver.py`
   - Módulos auxiliares si son necesarios
   - README con instrucciones de ejecución
2. **Informe técnico (máximo 10 páginas):**

    - Descripción del modelado del espacio de estados
    - Explicación detallada de las heurísticas
    - Análisis comparativo de algoritmos con tablas y gráficos
    - Casos de prueba con diferentes configuraciones
    - Conclusiones sobre qué algoritmo/heurística funciona mejor
3. **Demo:**
    - Video corto (3-5 minutos) mostrando el funcionamiento
    - Incluir al menos 3 configuraciones diferentes

Bonus

- **Visualización gráfica**: Interfaz con pygame o tkinter
- **Algoritmos adicionales**: Implementar IDA* o algún otro algoritmo
- **Generador de puzzles**: Crear puzzles con dificultad garantizada

Ejemplo de estado y movimiento

```
Estado inicial:          Movimiento: (0→3)          Estado resultante:
Tubo 0: [R, R, A, V]    Verter Tubo 0 a Tubo 3    Tubo 0: [A, V]
Tubo 1: [A, V, V, R]    (se vierten 2 rojos)      Tubo 1: [A, V, V, R]
Tubo 2: [A, R, A, V]                              Tubo 2: [A, R, A, V]
Tubo 3: [ ]                                       Tubo 3: [R, R]
Tubo 4: [ ]                                       Tubo 4: [ ]

Donde: R=Rojo, A=Azul, V=Verde
```

Recomendaciones

- Comenzar con configuraciones pequeñas (5 tubos, 3 colores)
- Implementar primero BFS para verificar corrección
- Probar las heurísticas en múltiples configuraciones antes de sacar conclusiones

Fecha de entrega

21 de octubre de 2025
