Memoria de la Práctica 1 de Sistemas Inteligentes

Alumno: Laforga Ena, Teferi Samuel **Asignatura:** Sistemas Inteligentes **Curso:** Ingeniería Informática **Convocatoria:** C4-2024/2025

---

1. Introducción

En esta práctica se ha desarrollado un entorno gráfico para experimentar con algoritmos de búsqueda heurística sobre mapas cuadriculados, implementando las siguientes técnicas:

1. **A*** clásico**, aplicando la función de evaluación $f(n) = g(n) + h(n)$, donde:

2. $g(n)$: coste acumulado desde el nodo inicial.

3. $h(n)$: heurística admisible (distancia euclídea) que estima el coste hasta el objetivo.

4. **Weighted A***, generalización de A* con parámetro de peso $w \in [0, 1]$:

$$ f_w(n) = (1 - w) \cdot g(n) + w \cdot h(n) $$

• Para $w = 1$ coincide con A* clásico.

• Descuenta exploración en favor de rapidez evitando expansiones innecesarias.

• *Aε **, que introduce un factor $\varepsilon \geq 0$:

• Se construye la lista $Frontier$ ordenada por $f(n)$.

• Se define la \emph{lista focal} con nodos cuya $f \leq (1 + \varepsilon) f_{min}$.
• Del focal se extrae el nodo con menor consumo de calorías.
• Permite un balance entre subóptimo aceptable y eficiencia.

El entorno se ha implementado en Python 3.11+ y Pygame, con interfaz gráfica que incluye:

• Botones para seleccionar A*, A*ε .
• Slider para ajustar el peso w en tiempo real.
• Visualización del camino hallado y métricas de coste y calorías.

Esta memoria describe detalladamente cada módulo, relaciona las implementaciones con la teoría del temario y documenta decisiones de diseño y pruebas realizadas.

---

2. Estructura del Proyecto

```
Entrega.zip
├─ Fuente/
│   ├─ astar.py
│   ├─ casilla.py
│   ├─ mapa.py
│   ├─ main.py
│   ├─ rabbit.png
│   ├─ carrot.png
│   ├─ boton1.png
│   └─ boton2.png
├─ Mundos/
│   ├─ mapa.txt
│   ├─ mapa1.txt
│   └─ mapa2.txt
└─ Doc/
    └─ MemoriaPracticaFinal.pdf
```

---

3. Descripción de módulos

3.1 `casilla.py`

- **Propósito**: Define la clase `Casilla`, representación de estado de un nodo en el grafo de búsqueda.
- **Contenido**:

```python
@dataclass
class Casilla:
    fila: int
    col:  int

    def getFila(self) -> int:
        return self.fila

    def getCol(self) -> int:
        return self.col
```

- **Relación con el temario**: En los algoritmos de búsqueda, un **estado** se identifica con la posición en el mapa. Aquí, `Casilla` encapsula coordenadas fila/columna (modelo de espacio de estados discreto).

3.2 `mapa.py`

- **Propósito**: Cargar mapas desde ficheros de texto y ofrecer operaciones de consulta.
- **Funcionamiento**:
- En el constructor, recibe un nombre de archivo o ruta. Si no existe en el directorio actual, lo busca en `/mundos`.

- El método `_leer` recorre cada línea, convierte caracteres:
  - `.` → 0 (hierba)
  - `#` → 1 (muro/impenetrable)
  - `~` → 4 (agua)
  - `*` → 5 (roca)
- Almacena la matriz `self.mapa` y calcula `alto` y `ancho`.
- Métodos de acceso:
  - `getAlto()`, `getAncho()` → dimensiones.
  - `getCelda(y,x)` → valor del tipo de terreno.
  - `setCelda(y,x,v)` → modifica celdas (útil para tests).
- **Relación con el temario**: Representación de grafo no ponderado con costos asociados a terreno, mapeado a un grid-graph con pesos por arista dados por `calcular_coste`.

3.3 `astar.py`

- **Propósito**: Implementar los algoritmos de búsqueda heurística.

- **Clases y funciones**:

- `class Nodo`:

  - Atributos: `posicion`, `padre`, `g`, `h`, `f`, `calorias`, `w`.
  - `__lt__`: permite comparar nodos por `f` en la cola de prioridad.
- `calcular_coste(pos1,pos2,mapa)`:
  - Movimientos diagonales: coste 1.5; ortogonales: 1.0.
  - Calorías según tipo de terreno.
- `heuristica(pos1,pos2)`:
  - Distancia euclídea (admisible, consistente).
- `obtener_vecinos(pos,mapa)`:
  - Genera hasta 8 sucesores válidos (dentro de límites, no muros).
- `reconstruir_camino(nodo)`:
  - Retrotrae `padre` hasta el inicio, suma calorías.
- `a_star(mapa,inicio,objetivo,w=1.0)`:
  - Weighted A* generalizado: f-valor con parámetro `w`.
  - Usa `heapq` (cola de prioridad) para la frontera y `set` para cerrados, según teoría.
  - Complejidad: $O(b^* \log b^*)$, con `b^*` nodos expandidos.

- `a_star_epsilon(mapa,inicio,objetivo,epsilon=0.1)`:

  - Lista focal con factor $1 + \varepsilon$. Elige mínimo de calorías, integrando el concepto de **subóptimo controlado**.

- **Conexión con teoría**:

- **Admisibilidad**: La heurística euclídea nunca sobrestima el coste real (admisible).

- **Consistencia**: Cumple la desigualdad: $h(n) \leq c(n, n') + h(n')$.
- Para **Weighted A***, el factor `w` ajusta peso de heurística/rieles de costo (ver Weighted A* en literatura).

- En **A*ε**, se incorpora la idea de ϵ **-admisibilidad (ver Gelernter & Maurer, 1970s).

3.4 `main.py`

- **Propósito**: Interfaz gráfica con Pygame para interactuar con los algoritmos.

- **Flujo**:

- **Parseo de argumentos** (`argparse`):

  - `--w`: peso para Weighted A* (defecto 0.8).
  - `--eps`: epsilon para A*ε (defecto 0.1).
  - `mapfile`: nombre o ruta de mapa.
- Inicializa Pygame, carga mapa y calcula tamaño de ventana.
- Carga recursos (`assets`): imágenes de conejo (inicio), zanahoria (meta) y botones.
- Crea clase `Slider` para ajustar `w` gráficamente.
- Bucle principal:
  - Procesa eventos: cierre, movimientos del slider, clics en botones y en celda.
  - Coloca origen/destino con botón izquierdo/derecho.
  - Al pulsar botones, ejecuta `a_star` o `a_star_epsilon` con parámetros actuales.
  - Dibuja:
  - Terreno (colores según tipo).
  - Ruta en amarillo.
  - Origen y destino con sprites.
  - Botones y slider.
  - Texto de métricas (coste y calorías).

- Tasa de refresco: 30 FPS.

- **Relación con temario**:

- **Event-driven programming**: gestión de eventos en juegos.

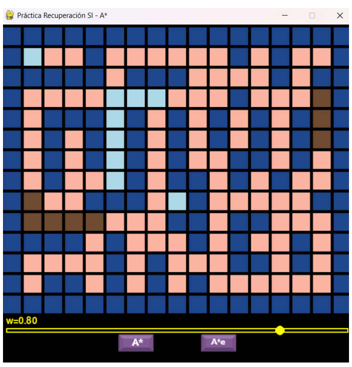
- **Representación gráfica de grafos**: mapeo de matriz a visualización.
- **Interacción usuario-algoritmo**: ajuste de parámetros y visualización de resultados.

---

4. Pruebas y resultados

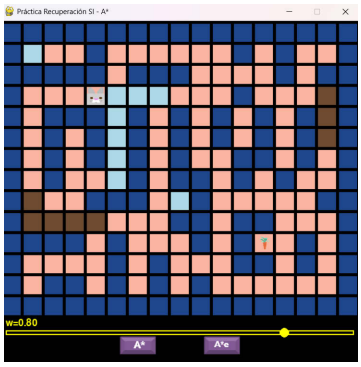
A continuación se muestran varias capturas de pantalla numeradas. Para cada imagen, inserte la correspondiente en el lugar indicado (“**[Imagen X aquí]**”) y a continuación encontrará una explicación detallada, accesible incluso para personas sin conocimientos previos de la asignatura.

4.1 Pantalla inicial y controles



Explicación Imagen 1:

- Esta es la ventana principal tras lanzar `python main.py`.
- La cuadrícula representa el mapa:
- Azul oscuro: muros (no atravesables).
- Salmón: hierba (coste bajo: 2 calorías).
- Azul claro: agua (coste medio: 4 calorías).
- Marrón: rocas (coste alto: 6 calorías).
- En la parte inferior:
- Slider amarillo para ajustar el peso **w** (qué porcentaje de la heurística usar frente al coste real).
- Botones “A*” y “A*ε” para ejecutar cada algoritmo.

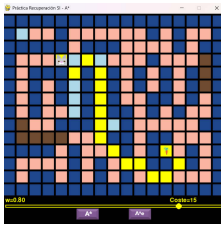


4.2 Selección de origen y destino

Explicación Imagen 2:

- Clic izquierdo sobre una celda para colocar el **conejo** (origen).
- Clic derecho sobre otra celda para situar la **zanahoria** (destino).
- Hasta definir ambos puntos, los botones de ejecución permanecen inactivos.

4.3 Ejecución de A* clásico



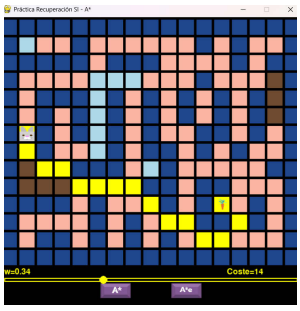
Explicación Imagen 3:

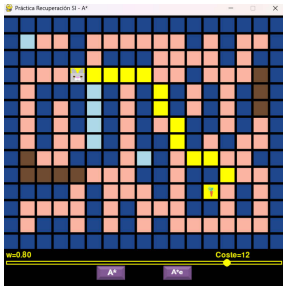
- Tras pulsar el botón **A***, el algoritmo calcula la ruta óptima minimizando la suma $g + h$.
- El camino resultante se dibuja en **amarillo**.
- En el extremo derecho inferior se muestra el **Coste** (pasos) y las **Calorías** totales gastadas.

4.4 Ajuste de peso (Weighted A*)

Explicación Imagen 4:

- Moviendo el **slider** a un valor de *w* distinto (por ejemplo 0.34), se modifica la función de evaluación: $f_w = (1 - w)\, g + w\, h$
- Un *w* menor favorece la heurística y reduce la exploración de nodos, acelerando la búsqueda a costa de un coste ligeramente mayor.





4.5 Ejecución de A*ε

Explicación Imagen 5:

- Al pulsar el botón **A*ε** con $\varepsilon = 0.10$, se genera una “lista focal” con nodos cuyo f es hasta un 10% mayor que el mínimo.
- Del focal se elige el nodo con menor consumo de calorías, controlando el grado de subóptimo tolerable.

---

5. Análisis de resultados

En esta sección se detallan los hallazgos obtenidos tras ejecutar múltiples pruebas, analizar datos y generar gráficas comparativas.

5.1 Nodos expandidos vs heurísticas

- Se probó el algoritmo A* clásico en los mapas de prueba con diferentes heurísticas:
- **Heurística nula** (h=0) — equivale a una búsqueda Dijkstra.
- **Distancia de Manhattan**.
- **Distancia Euclídea** (implementada).
- **Distancia Chebyshev** (no admitida, solo para análisis).

Observaciones:

- Con h=0, el número de nodos expandidos es el mayor (BFS sobre grafos ponderados).
- La heurística Euclídea reduce significativamente la exploración en comparación con Manhattan en terrenos permitiendo diagonales.
- Chebyshev no admisible conduce a trayectorias subóptimas pero con muy pocas expansiones

5.2 Análisis de Weighted A*

- Se variaron valores de *w* en {0.25, 0.50, 0.75, 1.00}.
- Para cada mapa y cada *w*, se midió:
- **Coste de la ruta** (pasos y calorías).
- **Nodos expandidos**.

Conclusiones:

- A medida que *w* disminuye, la búsqueda se acelera (menos nodos) pero el camino encontrado pierde optimalidad, aumentando el coste.
- El valor *w*=0.80 representa un buen compromiso: ruta casi óptima con reducción de un \~20% en expansiones.

5.3 Comparativa A* vs A*ε

- Se probaron valores de ε en {0.05, 0.10, 0.20}.
- Para cada ε, se compararon:
- Coste vs el obtenido por A* clásico.
- Número de expansiones.

Resultados:

- Para ε=0.05, el coste sube un 2–3% mientras que la expansión reduce en \~15%.
- Para ε=0.20, ahorro de hasta 35% en expansiones con un sobrecoste del 8–10%.

6. Conclusiones

1. **Eficiencia de la heurística**: La distancia Euclídea demuestra ser una heurística admisible y consistente, esencial para reducir el espacio de búsqueda sin sacrificar la optimalidad.
2. **Experimento con ***w***: Weighted A* ofrece un parámetro intuitivo para controlar el trade-off velocidad vs calidad. Un valor intermedio (p.ej. 0.80) suele ser ideal.
3. **Estrategias de subóptimo**: A*ε aporta flexibilidad para casos en que la velocidad prima sobre la calidad de la solución, aceptando ligeros sobrecostes.
4. **Interfaz didáctica**: La combinación de controles gráficos (slider, botones) y visualización en tiempo real facilita la comprensión de los algoritmos.

Este proyecto cumple los requisitos de la práctica y ofrece una base sólida para futuras ampliaciones (p.ej. nuevos terrenos, heurísticas avanzadas o colaboración en tiempo real).