Tarea 1 - Laberinto Saltarín

La implementación creada busca resolver el problema del “Laberinto Saltarín”, permitiendo cargar múltiples laberintos desde un mismo archivo de texto. Para la resolución del problema se utilizan algoritmos de Búsqueda en profundidad (DFS) y Costo Uniforme (UCS). La interfaz gráfica esta implementada con la librería ‘Pygame’, la cual consiste en un menú en el cual se puede seleccionar el laberinto y el algoritmo con el cual se desea resolver. Luego se Visualiza el laberinto, indicando con colores la celda de inicio y la celda objetivo, una vez iniciada la ejecución se visualizan los caminos recorridos y en caso de llegar se resalta el camino elegido. Se guarda el historial de pasos, para así poder recorrer los pasos ejecutados de forma inversa.

Estructura del Código

Maze Class:

- Almacena las dimensiones (m, n), la posición inicial (start_pos), la posición final (goal_pos) y la propia grilla (grid).
- get_jump_value(pos): Devuelve el número en la celda especificada.
- is_valid(pos): Comprueba si una posición está dentro de los límites de la grilla.
- get_neighbors(pos): Calcula las posiciones alcanzables desde pos según el valor de salto, respetando los límites y la regla de movimiento horizontal/vertical. Devuelve una lista de vecinos válidos.

SolverState Class:

- current_node: La celda que se está procesando en este paso.
- path_to_current: La secuencia de celdas desde el inicio hasta current_node.
- frontier: Un set de posiciones que están en la frontera (lista de espera) del algoritmo.
- visited: Un set de posiciones que ya han sido exploradas.
- final_path: La ruta completa de la solución si se ha encontrado.

Clases de Solvers:

Solver: Proporciona la estructura común para todos los solvers.

- __init__(maze): Inicializa con el laberinto a resolver.
- history: Una lista para almacenar objetos SolverState, registrando cada paso.
- current_step_index: Índice actual en el history para la visualización.
- solution_path, message, visited: Almacenan el resultado final y el estado general.
- step(): Método abstracto que debe ser implementado por las subclases para realizar un paso del algoritmo específico.
- get_current_state(): Devuelve el SolverState correspondiente al current_step_index.
- next_step(), prev_step(): Navegan por el historial.
- reset(): Reinicia el solver a su estado inicial.

DFSSolver(Solver): Implementa la búsqueda en profundidad.

- _initialize_search(): Crea una stack con el nodo inicial y su camino. Marca el inicio como visitado.
- step():

    1. Si la pila está vacía, termina.
    2. Saca un nodo (current_pos, path) de la pila.
    3. Marca current_pos como visitado.
    4. Registra el estado antes de explorar vecinos.
    5. Si es el nodo objetivo, guarda la solución, registra el estado final y termina.
    6. Obtiene los vecinos válidos.
    7. Para cada vecino no visitado, lo añade a la pila con el camino extendido.

UCSSolver(Solver): Implementa la Búsqueda de Costo Uniforme.

- _initialize_search(): Crea una queue con el nodo inicial y su camino. Marca el inicio como visitado al añadirlo a la cola.
- step():

    1. Si la cola está vacía, termina.
    2. Saca un nodo (current_pos, path) de la cola.
    3. Registra el estado antes de explorar vecinos.
    4. Si es el nodo objetivo, guarda la solución, registra el estado final y termina.
    5. Obtiene los vecinos válidos.
    6. Para cada vecino no visitado:
        - Lo marca como visitado.
        - Lo añade a la cola con el camino extendido.

Parsing de Entrada (parse_input_file): Valida el formato y secciona el archivo de entrada para guardar las características de cada laberinto.

Componentes GUI:

Button Class: Clase simple para crear botones rectangulares con texto.

calculate_grid_params(m, n): Calcula el tamaño óptimo para que el laberinto m x n quepa en el área de visualización disponible, manteniendo las celdas cuadradas.

draw_menu: Dibuja la pantalla del menú principal: título, selector de laberinto y botones: selección de algoritmo (DFS/UCS) y ‘Start’.

draw_solver_view: Dibuja la vista del laberinto, con los botones (start, pause, anterior, siguiente y menú) respectivos.

Ejemplo Input / output:

1.- Input: 2 2 0 0 1 1 / 1 1 / 1 0, output: DFS = 2 movimientos, UCS = 2 movimientos

2.- 3 3 0 0 2 2 / 0 1 1 / 1 1 1 / 1 1 0, output: DFS = UCS = No hay solución

3.- 4 1 0 0 2 0 / 2 / 1 / 0 / 1, output: DFS = 1 movimientos, UCS = 1 movimientos

4.- 4 4 0 0 3 3 / 1 1 1 3 / 1 1 1 1 / 3 1 1 1 / 1 1 1 0, output: DFS = 6 movimientos, UCS = 4 movimientos

5.- 5 5 0 0 4 4 / 2 1 1 1 1 / 1 3 1 1 1 / 1 1 4 1 1 / 1 1 1 1 2 / 1 1 1 1 0, output: DFS = 9 movimientos, UCS = 7 movimientos

6.- 3 5 0 0 2 4 / 1 1 1 1 2 / 1 2 1 3 1 / 1 1 1 1 0 output: DFS = 9 movimientos, UCS = 5 movimientos