Memoria del Proyecto: Visualización de Algoritmos de Búsqueda de Caminos

1. Introducción

El proyecto consiste en el desarrollo de una aplicación gráfica utilizando Tkinter en Python, cuyo propósito es implementar y visualizar en tiempo real dos algoritmos de búsqueda de caminos: **A*** y **Dijkstra**. La aplicación permite al usuario configurar diversos parámetros (como la elección de heurística, la posibilidad de movimientos diagonales y la modificación del terreno) para observar cómo varía la exploración y la generación de rutas en un entorno de cuadrícula.

2. Objetivos

- **Comparación y análisis de algoritmos:**
  Permitir la comparación entre A* y Dijkstra, resaltando las diferencias en la búsqueda al incorporar (o no) una heurística.
- **Visualización interactiva:**
  Desarrollar una interfaz gráfica que muestre de forma progresiva el proceso de búsqueda, permitiendo visualizar nodos explorados y la expansión de la frontera en tiempo real.
- **Flexibilidad en la configuración:**
  Ofrecer opciones para seleccionar la función heurística (Manhattan o Euclidiana) y definir si se permiten movimientos diagonales, separando de forma explícita la lógica de la heurística de la selección de direcciones de movimiento.
- **Interactividad y personalización:**
  Permitir al usuario definir puntos de inicio y meta, así como modificar el terreno (por ejemplo, estableciendo áreas de "agua", "pared", etc.) para observar cómo estos cambios afectan la búsqueda.

3. Descripción General y Diseño

3.1. Funcionalidades Clave

- **Selección de algoritmos:**
  La aplicación ofrece un menú para elegir entre A* y Dijkstra. Mientras que A* utiliza una heurística para orientar la búsqueda, Dijkstra opera únicamente en base al costo acumulado (equivalente a utilizar una heurística nula).
- **Configuración de heurística:**
  Se permite seleccionar entre heurística Manhattan y Euclidiana. Es importante destacar que la heurística solo afecta la estimación del costo restante y no determina las direcciones de movimiento.
- **Control de direcciones:**
  A través de un checkbox, el usuario puede activar o desactivar los movimientos diagonales.

Esto es independiente de la función heurística elegida, asegurando que la dirección de movimiento se configure exclusivamente a través de esta opción.

- **Visualización en tiempo real:**
  Durante la búsqueda, se actualiza la cuadrícula para reflejar:
  - **Nodos expandidos:** Se marcan en color gris claro.
  - **Nodos añadidos a la frontera:** Se muestran en color cian.
  - **Ruta final:** Una vez encontrada, se destaca en color naranja, excepto la celda meta, para no tapar su identificación.

3.2. Estructura del Código

- **Importaciones y Configuración:**
  Se utilizan librerías estándar como Tkinter para la interfaz gráfica, `heapq` para la gestión de la cola de prioridad, y NumPy para cálculos relacionados con la heurística Euclidiana. La configuración del terreno (colores y pesos) se define en un diccionario, facilitando la personalización.
- **Clase `PathfindingApp`:**
  Esta clase centraliza toda la funcionalidad de la aplicación. Entre sus métodos destacan:
  - `draw_grid`: Dibuja la cuadrícula completa según el estado actual.
  - `on_click`: Permite la interacción del usuario para definir puntos de inicio, meta y modificar el terreno.
  - `find_path` y `calculate_path`: Implementan la lógica de búsqueda utilizando A* o Dijkstra y realizan la visualización en tiempo real del proceso.
  - Métodos para configurar opciones (algoritmo, heurística, movimiento diagonal, etc.) y para actualizar la interfaz según las interacciones.
- **Separación de Heurística y Movimiento:**
  Se garantiza que la selección de la función heurística no influya en las direcciones de movimiento. Así, el checkbox "Permitir diagonales" controla únicamente si se consideran o no movimientos en diagonal, independientemente de si se utiliza Manhattan o Euclidiana como función heurística.

4. Desarrollo e Implementación

4.1. Proceso de Búsqueda y Visualización

- **Algoritmo A* vs. Dijkstra:**
  - **A***: Se calcula la prioridad de cada nodo como la suma del costo acumulado y el valor de la heurística hasta la meta. Esto permite dirigir la búsqueda hacia el objetivo, reduciendo la exploración en áreas no prometedoras.
  - **Dijkstra:** Se implementa definiendo la heurística como cero, lo que obliga al algoritmo a evaluar todos los nodos en función del costo acumulado, generando una exploración más uniforme.
- **Visualización en Tiempo Real:**
  Se incorporan llamadas a `time.sleep(DELAY)` y `self.canvas.update()` para

pausar y actualizar la interfaz gráfica en cada paso del algoritmo. Esto permite observar cómo se expanden los nodos y se agregan vecinos a la frontera en cada iteración.

4.2. Interfaz de Usuario

La interfaz fue diseñada para ser intuitiva y permitir:

- Seleccionar el algoritmo y la heurística a través de menús desplegables.
- Activar o desactivar el movimiento diagonal mediante un checkbox.
- Definir el terreno y los puntos clave (inicio y meta) mediante clics en la cuadrícula.
- Visualizar el proceso de búsqueda con indicadores gráficos (colores específicos para cada tipo de celda y estado).

5. Resultados y Análisis

La implementación permite observar claramente:

- **La eficiencia de A*** cuando se utiliza una heurística adecuada, ya que la búsqueda se concentra en la dirección de la meta, reduciendo el número de nodos evaluados.
- **La exploración más amplia de Dijkstra,** la cual resulta en una búsqueda radial cuando no se dispone de una estimación del costo restante.
- **El impacto del movimiento diagonal:** Al permitir movimientos diagonales, la búsqueda puede acortar rutas, pero también puede generar caminos menos naturales en ciertos escenarios. La separación de esta funcionalidad asegura que la función heurística se evalúe de manera independiente.

6. Conclusiones y Sugerencias

Flexibilidad y didáctica:

La aplicación cumple con el objetivo de servir como herramienta didáctica para visualizar y comparar algoritmos de búsqueda. La capacidad de modificar parámetros en tiempo real permite entender de manera práctica las implicaciones de cada elección (algoritmo, heurística, movimientos).

Claridad en la separación de funciones:

Al diferenciar la selección de la heurística de las direcciones de movimiento, se garantiza una mayor precisión en el comportamiento del algoritmo, evitando confusiones en la configuración.