TECNOLÓGICO
NACIONAL DE MÉXICO

Inteligencia Artificial – Puzzle 8

Alumnos:

- Lara Beltran Jorge Alberto
- Pacheco Perez Diego

Docente:

Zuriel Dathan Mora Felix

Grupo:

09-10

1. Introducción

Este documento describe el funcionamiento del Proyecto puzzle8 en Python utilizando el algoritmo A* y una interfaz gráfica desarrollada con Tkinter.

2. Importaciones

```
import tkinter as tk
from tkinter import messagebox
import heapq
import time
```

- **tkinter**: para construir la interfaz gráfica.
- **messagebox**: para mostrar mensajes emergentes.
- **heapq**: para manejar la cola de prioridad en el algoritmo A*.
- **time**: para medir el tiempo de ejecución.

3. Configuración del Puzzle Final

Se define la configuración meta (objetivo) del puzzle como una matriz 3x3 ordenada:

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

4. Funciones de Utilería

findZero(puzzle): Encuentra la posición del espacio vacío (0).

```
def findZero(puzzle):
    for i in range(3):
        for j in range(3):
            if puzzle[i][j] == 0:
                return i, j
```

movePice(puzzle, movement): Genera un nuevo puzzle con el movimiento solicitado ('Arriba', 'Abajo', 'Izquierda', 'Derecha').

```
def movePice(puzzle, movement):
    new_puzzle = [row.copy() for row in puzzle]
    i, j = findZero(new_puzzle)
    if movement == "Arriba" and i > 0:
        new_puzzle[i][j], new_puzzle[i-1][j] = new_puzzle[i-1][j],
new_puzzle[i][j]
    elif movement == "Abajo" and i < 2:
        new_puzzle[i][j], new_puzzle[i+1][j] = new_puzzle[i+1][j],
new_puzzle[i][j]
    elif movement == "Izquierda" and j > 0:
        new_puzzle[i][j], new_puzzle[i][j-1] = new_puzzle[i][j-1],
new_puzzle[i][j]
    elif movement == "Derecha" and j < 2:
        new_puzzle[i][j], new_puzzle[i][j+1] = new_puzzle[i][j+1],
new_puzzle[i][j]
    else:
        return None
    return new_puzzle
```

5. Clase Node

La clase Node representa un estado dentro de la búsqueda A*.
Atributos:
- puzzle: estado actual.
- movimiento: movimiento aplicado para llegar al nodo.
- costo: número de movimientos desde el inicio.
- heuristica: distancia estimada hasta la meta.
- parent: referencia al nodo anterior.

El método __lt__ permite ordenar nodos en la cola de prioridad.

```
class Node:
    def __init__(self, puzzle, movimiento, costo , heuristica, parent):
        self.puzzle = puzzle
        self.movimiento = movimiento
        self.costo = costo
        self.heuristica = heuristica
        self.parent = parent

    def __lt__(self, other):
        return (self.costo + self.heuristica) < (other.costo +
other.heuristica)
```

6. Heurística

CalcularHeuristica(puzzle): Calcula la suma de las distancias de Manhattan de cada ficha hasta su posición final.

```
def CalcularHeuristica(puzzle):
    heuristica = 0
    for i in range(3):
        for j in range(3):
            if puzzle[i][j] != 0:
                i2, j2 = getPosicion(puzzle[i][j])
                heuristica += abs(i - i2) + abs(j - j2)
    return heuristica
```

getPosicion(valor): Devuelve la posición correcta de cada valor en el estado meta.

```
def getPosicion(valor):
    for i in range(3):
        for j in range(3):
            if puzzleFinal[i][j] == valor:
                return i,j
```

7. Algoritmo A*

El algoritmo utiliza una cola de prioridad para explorar nodos en orden de menor costo + heurística. Se generan nuevos estados moviendo el 0 en las cuatro direcciones posibles. Cuando se encuentra el estado final, se reconstruye el camino hacia atrás mediante los nodos padre.

```
def algoritmo_a_star(puzzle):
    nodosVisitados = set()
    cola = []
    heapq.heappush(cola, Node(puzzle, "", 0, CalcularHeuristica(puzzle),
None))

    while cola:
        actual = heapq.heappop(cola)
        if actual.puzzle == puzzleFinal:
            break
        nodosVisitados.add(str(actual.puzzle))
        for movimiento in ["Arriba","Abajo","Izquierda","Derecha"]:
            siguiente = movePice(actual.puzzle, movimiento)
            if siguiente and str(siguiente) not in nodosVisitados:
                heapq.heappush(cola, Node(siguiente, movimiento,
actual.costo+1, CalcularHeuristica(siguiente), actual))
```

```
    recorrido = []
    while actual:
        recorrido.append(actual)
        actual = actual.parent
    return recorrido[::-1]
```

8. Verificación de Resolubilidad

La función es_resolvible convierte el puzzle en una lista lineal sin el 0, calcula el número de inversiones (pares desordenados) y determina si el puzzle es resoluble. Un puzzle es resoluble si el número de inversiones es par.

```
def es_resolvible(puzzle):
    plano = [num for fila in puzzle for num in fila if num != 0]
    inversiones = sum(1 for i in range(len(plano)) for j in
range(i+1,len(plano)) if plano[i] > plano[j])
    return inversiones % 2 == 0
```

9. Interfaz Gráfica (Tkinter)

La clase PuzzleUI gestiona la interfaz:
- mostrarPuzzle(): actualiza la cuadrícula visual del tablero.
- resolver(): ejecuta el algoritmo A* y muestra la solución.
- animar(): anima paso a paso la secuencia de movimientos usando after().

```
 def mostrarPuzzle(self):
        for i in range(3):
            for j in range(3):
                val = self.puzzle[i][j]
                self.labels[i][j].config(text="" if val==0 else str(val))

    def resolver(self):
        if not es_resolvible(self.puzzle):
            messagebox.showerror("Error","🚨 Este puzzle NO tiene
solución. Intenta con otro orden. 🚨")
            return
        inicio = time.time()
        pasos = algoritmo_a_star(self.puzzle)
        duracion = time.time()-inicio
        self.animar(pasos,0)
        messagebox.showinfo("Listo!", f"Resuelto en {len(pasos)-1}
movimientos.\nTiempo: {duracion:.3f} s")
```

```
    def animar(self, pasos, idx):
        if idx >= len(pasos): return
        self.puzzle = pasos[idx].puzzle
        self.mostrarPuzzle()
        self.root.after(500, lambda: self.animar(pasos, idx+1))
```

10. Ejecución del Programa

Se define un puzzle inicial, se crea la ventana principal de Tkinter y se ejecuta el ciclo principal con root.mainloop().

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


root = tk.Tk()
app = PuzzleUI(root, puzzleInicial)
root.mainloop()
```

11. Resumen

Este programa combina el algoritmo A* con una interfaz gráfica para resolver y visualizar el puzzle de 8 fichas, permite verificar si un puzzle se puede resolver, calcular una solución óptima y mostrarla.