TECNOLÓGICO NACIONAL DE MÉXICO

Inteligencia Artificial

Tarea 4:

Puzzle 8

Alumno:

Rementeria Medina Jesus Hector

Este proyecto consiste en por medio del algoritmo A* buscar la solución con menos movimientos posibles del juego del Puzzle 8

```python
def jugar_puzzle_8():  1 usage
    #Generar un tablero aleatorio
    inicio = generar_tablero_aleatorio()

    #Tablero inicial fijo
    #inicio = [
        #[5, 1, 3],
        #[7, 2, 8],
        #[4, 0, 6]
    #]

    #Objetivo final del tablero
    objetivo = [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 0]
    ]

    #Imprmir el tablero inicial
    print("Estado inicial:")
    imprimir_tablero(inicio)
```

Al iniciar el programa se lleva a cabo la creación de un tablero de forma aleatorio esto por medio de la generación de números pseudoaleatorios los cuales son introducidos a un Array simulando el tablero del juego con el numero 0 representando el vacío.

```python
#Genrar un tablero aleatorio
def generar_tablero_aleatorio():  1 usage
    tablero = [i for i in range(9)]
    while True:
        random.shuffle(tablero)
        matriz = [tablero[i:i + 3] for i in range(0, 9, 3)]
        if es_resoluble(matriz):
            return matriz

#Comprobar si el tablero tiene solucion
def es_resoluble(tablero):  1 usage
    plano = sum(tablero, [])
    inversions = sum(
        1 for i in range(len(plano)) for j in range(i + 1, len(plano)) if plano[i] and plano[j] and plano[i] > plano[j])
    return inversions % 2 == 0
```

El método “generar_tablero_aleatorio” genera una lista con los números del 0 al 8, luego revuelve los números para luego comprobar si el tablero tiene solución, si el tablero generado no puede ser resuelto lo vuelve a revolver hasta que logre encontrar uno con solución.

Se considera que un tablero tiene solución si se cumple la condición de tener un numero par de inversiones, se considera una inversión cuando número mayor aparece antes que otro menor, el 0 no es considerado al momento de calcular las inversiones.

```
solucion, tiempo_transcurrido = a_star(inicio, objetivo)
```

A continuación, se lleva a cabo la búsqueda de la solución haciendo uso del algoritmo de A*, se le manda al método el estado inicial y cual es el objetivo al que se busca llegar

```
def a_star(inicio, objetivo):  1 usage
    #Inicia el temporizador
    inicio_tiempo = time.time()
    #Cola de proridad
    prioridad = []
    #
    estado_inicial = Estado(inicio, heuristica=distancia_manhattan(inicio, sum(objetivo, [])))
    #Agregar estado inicial a la cola
    heapq.heappush( *args: prioridad, estado_inicial)
    #Estados visitados
    visitados = set()
```

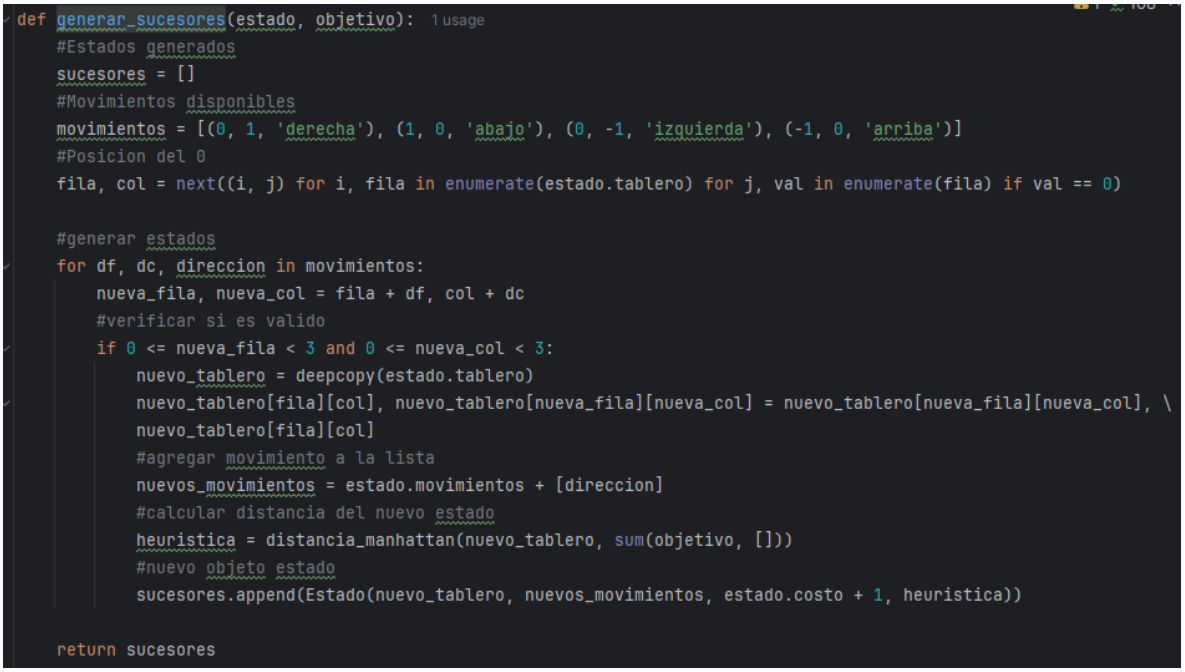
El método inicia un temporizador para calcular cuánto tiempo se tardó en encontrar la solución, se crea la cola de prioridad y se crea el estado inicial y se calcula la heurística usando la distancia de manhattan

```python
#Algoritmo distancia manhattan
def distancia_manhattan(tablero, objetivo):  2 usages
    distancia = 0
    #Recorrer el tablero
    for i in range(3):
        for j in range(3):
            #Ignorar el 0
            if tablero[i][j] != 0:
                #encontrar posicion correcta
                fila_obj, col_obj = divmod(objetivo.index(tablero[i][j]), 3)
                #calcular distancia
                distancia += abs(fila_obj - i) + abs(col_obj - j)
    return distancia
```

El algoritmo de distancia manhattan recorre todo el tablero calculando la distancia entre los números en su estado inicial y su objetivo para calcular la distancia

```python
    while prioridad:
        estado_actual = heapq.heappop(prioridad)

        if estado_actual.tablero == objetivo:
            fin_tiempo = time.time()
            tiempo_transcurrido = fin_tiempo - inicio_tiempo
            return estado_actual.movimientos, tiempo_transcurrido

        #Marcar estado como visitado
        visitados.add(estado_actual)

        #Posibles movimientos
        for sucesor in generar_sucesores(estado_actual, objetivo):
            if sucesor not in visitados:
                heapq.heappush( *args: prioridad, sucesor)

    #No se encontro solucion
    return None, None
```

Mientras la lista de prioridad tenga elementos se exploran los estados posibles, se extrae el estado con menor costo y se comprueba si se encontró la solución, si no se encontró la solución se agrega el estado actual al conjunto de visitados para que no se intente volver a explorar el mismo estado.



```
def generar_sucesores(estado, objetivo):  1 usage
    #Estados generados
    sucesores = []
    #Movimientos disponibles
    movimientos = [(0, 1, 'derecha'), (1, 0, 'abajo'), (0, -1, 'izquierda'), (-1, 0, 'arriba')]
    #Posicion del 0
    fila, col = next((i, j) for i, fila in enumerate(estado.tablero) for j, val in enumerate(fila) if val == 0)

    #generar estados
    for df, dc, direccion in movimientos:
        nueva_fila, nueva_col = fila + df, col + dc
        #verificar si es valido
        if 0 <= nueva_fila < 3 and 0 <= nueva_col < 3:
            nuevo_tablero = deepcopy(estado.tablero)
            nuevo_tablero[fila][col], nuevo_tablero[nueva_fila][nueva_col] = nuevo_tablero[nueva_fila][nueva_col], \
            nuevo_tablero[fila][col]
            #agregar movimiento a la lista
            nuevos_movimientos = estado.movimientos + [direccion]
            #calcular distancia del nuevo estado
            heuristica = distancia_manhattan(nuevo_tablero, sum(objetivo, []))
            #nuevo objeto estado
            sucesores.append(Estado(nuevo_tablero, nuevos_movimientos, estado.costo + 1, heuristica))

    return sucesores
```

Se generan los estados sucesores donde intentara mover el vacío (numero 0) a las direcciones posibles, si el sucesor no se encuentra visitado lo agrega a la cola de prioridad ordenándolo según el costo y la heurística, de esta forma el algoritmo va visitando los estados con menor costo hasta encontrar la solución, mientras esto se realiza los movimientos necesarios para llegar a la solución son agregados a la lista de estado_actual, para al final regresar cuales fueron los mejores movimientos encontrados y cuánto tiempo se tardaron en encontrar

```python
if solucion:
#Imprimir los movimientos realizados y el estado de los tableros
    estado_actual = deepcopy(inicio)
    #Imprimir los movimientos realizados
    for i, movimiento in enumerate(solucion, 1):
        fila, col = next((i, j) for i, fila in enumerate(estado_actual) for j, val in enumerate(fila) if v
        if movimiento == 'derecha':
            estado_actual[fila][col], estado_actual[fila][col + 1] = estado_actual[fila][col + 1], \
            estado_actual[fila][col]
        elif movimiento == 'izquierda':
            estado_actual[fila][col], estado_actual[fila][col - 1] = estado_actual[fila][col - 1], \
            estado_actual[fila][col]
        elif movimiento == 'arriba':
            estado_actual[fila][col], estado_actual[fila - 1][col] = estado_actual[fila - 1][col], \
            estado_actual[fila][col]
        elif movimiento == 'abajo':
            estado_actual[fila][col], estado_actual[fila + 1][col] = estado_actual[fila + 1][col], \
            estado_actual[fila][col]
        print(f"Movimiento {i}: {movimiento}")
        imprimir_tablero(estado_actual)
    print(f"Solución encontrada en {len(solucion)} movimientos.")
    print(f"Tiempo transcurrido: {tiempo_transcurrido:.4f} segundos.")
```

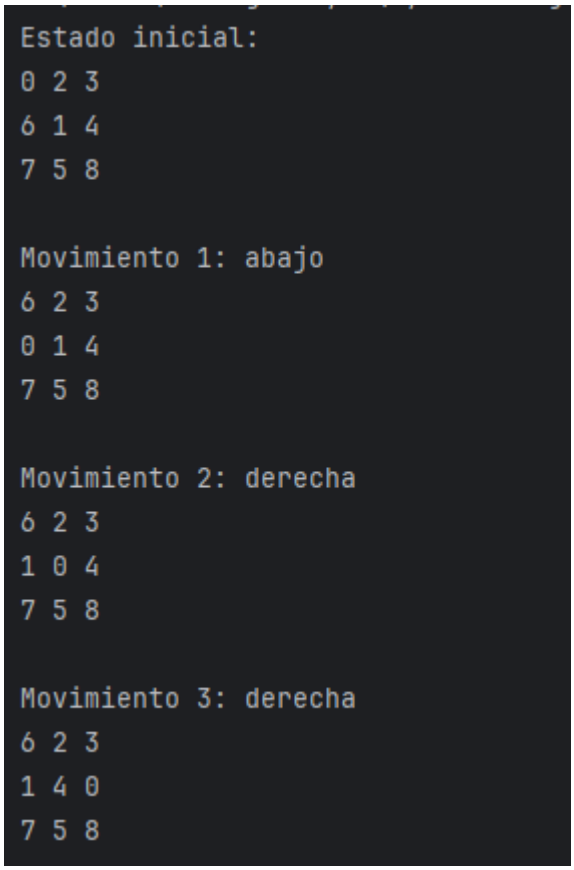
Ya con los movimientos encontrados se prosigue a mostrar cuales fueron estos, dentro del for se realiza una copia del estado inicial del tablero para luego sacar de la lista cual fue el primer movimiento y lo lleva a cabo para el tablero inicial para luego imprimirlo como el primer movimiento, después guarda el estado actual después de realizar el movimiento y continua llevando a cabo todos los movimientos, así va mostrando cada uno de los movimientos necesarios y la posición de los numero cuando estos se realizaron hasta llegar al objetivo, por último se muestra la cantidad de movimientos y el tiempo transcurrido

A continuación, se realizarán algunas pruebas del funcionamiento del programa

```
Estado inicial:
4 8 5
2 7 6
0 1 3

Movimiento 1: derecha
4 8 5
2 7 6
1 0 3

Movimiento 2: arriba
4 8 5
2 0 6
1 7 3

Movimiento 3: derecha
4 8 5
2 6 0
1 7 3
```

```
Movimiento 24: derecha
1 2 3
4 0 6
7 5 8

Movimiento 25: abajo
1 2 3
4 5 6
7 0 8

Movimiento 26: derecha
1 2 3
4 5 6
7 8 0

Solución encontrada en 26 movimientos.
Tiempo transcurrido: 0.1717 segundos.
```

El tablero fue resuelto en 26 movimientos tomándole 0.1717 segundos en resolverlo, se muestra cual fue el tablero inicial y que movimientos se realizaron hasta llegar a la solución



```
Movimiento 12: izquierda
1 2 3
4 0 6
7 5 8

Movimiento 13: abajo
1 2 3
4 5 6
7 0 8

Movimiento 14: derecha
1 2 3
4 5 6
7 8 0

Solución encontrada en 14 movimientos.
Tiempo transcurrido: 0.0040 segundos.
```

Esta vez la solución se logró en 14 movimientos y solo tardo 0.0040 segundos en encontrarla