*Juego del 8 A**

Trabajo Corte 1 - Computación 2

Nombre: Mariana Rodríguez Pérez

Fecha: Marzo 2025

El juego del ocho es un rompecabezas deslizante que consiste en una cuadrícula de 3x3 donde se encuentran 8 fichas numeradas y una casilla vacía. El objetivo es mover las fichas, deslizando aquellas que se encuentran en la misma fila o columna que la casilla vacía, hasta ordenar los números de forma secuencial (de izquierda a derecha y de arriba hacia abajo), dejando la casilla vacía en la última posición.

El código se realizó en Python utilizando conceptos básicos de programación orientada a objetos. Se creó una clase llamada **Game** que:

- Inicializa el tablero con una matriz 3x3.
- Define métodos para visualizar el tablero usando **matplotlib**.
- Implementa funciones para encontrar la casilla vacía y validar si los movimientos (arriba, abajo, izquierda o derecha) son permitidos.
- Permite realizar los movimientos actualizando el estado del tablero y registra el historial de movimientos.

Esta estructura modular facilita la extensión y mejora del código, permitiendo una gestión clara de la lógica del juego y su visualización gráfica.

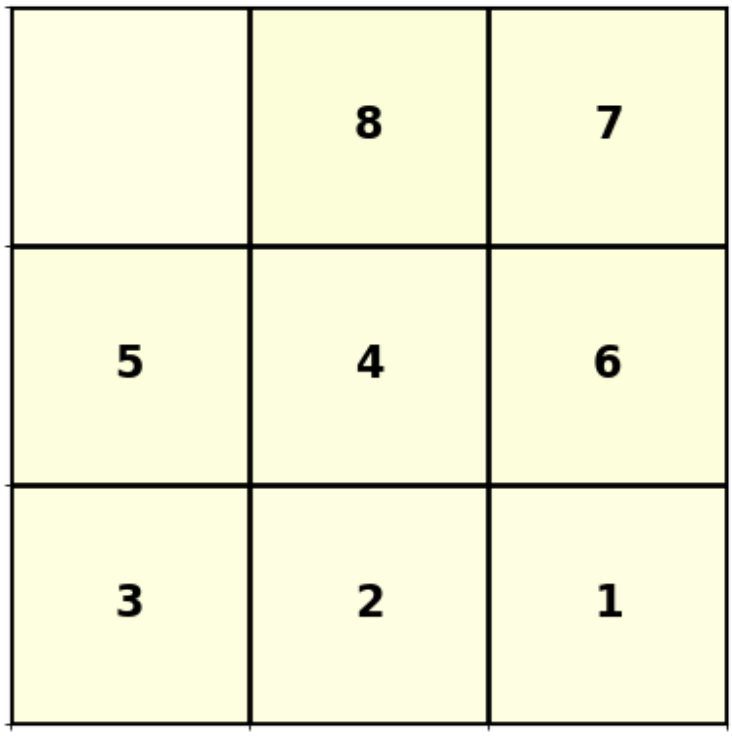
```
import heapq
import numpy as np
import matplotlib.pyplot as plt
import datetime

class Position:
    def __init__(self, x, y):
        self.x = x
        self.y = y

class Node:
    def __init__(self, stage, move, cost, heuristic):
        self.stage = stage
        self.move = move
        self.cost = cost  # g(n) -> Costo acumulado
        self.heuristic = heuristic  # h(n) -> Estimación del costo
restante

    def __lt__(self, other):
        return (self.cost + self.heuristic) < (other.cost +
```

```
other.heuristic)

class Game:
    GOAL = [[1, 2, 3],
            [4, 5, 6],
            [7, 8, None]]

    def __init__(self, initial_board):
        self.board = [row[:] for row in initial_board]
        self.position_empty_space = self.__find_empty_space_position()

    def __find_empty_space_position(self):
        for y in range(len(self.board)):
            for x in range(len(self.board[y])):
                if self.board[y][x] is None:
                    return Position(x, y)
        raise Exception("No se encontró un espacio vacío en el juego")

    def __copy(self):
        return Game([row[:] for row in self.board])

    def show(self):
        _, ax = plt.subplots()
        img_data = np.array([[0 if x is None else x for x in row] for
row in self.board])
        plt.imshow(img_data, cmap="YlGn", interpolation="nearest",
vmin=0, vmax=255)
        ax.set_xticks(np.arange(-0.5, len(self.board[0]), 1),
minor=True)
        ax.set_yticks(np.arange(-0.5, len(self.board), 1), minor=True)
        ax.grid(which="minor", color="black", linestyle="-",
linewidth=2)
        ax.set_xticks([])
        ax.set_yticks([])
        for i in range(len(self.board)):
            for j in range(len(self.board[0])):
                value = self.board[i][j]
                text = str(value) if value is not None else " "
                ax.text(j, i, text, ha='center', va='center',
fontsize=16, fontweight='bold')
        plt.show()

    def is_game_win(self):
        return self.board == self.GOAL

    def move(self, dx, dy):
        new_game = self.__copy()
        new_x, new_y = new_game.position_empty_space.x + dx,
new_game.position_empty_space.y + dy
        new_game.board[new_game.position_empty_space.y]
```

```
[new_game.position_empty_space.x] = new_game.board[new_y][new_x]
        new_game.board[new_y][new_x] = None
        new_game.position_empty_space = Position(new_x, new_y)
        return new_game

    def next_allowed_moves(self):
        moves = []
        if self.position_empty_space.y > 0:
            moves.append(("UP", self.move(0, -1)))
        if self.position_empty_space.y < len(self.board) - 1:
            moves.append(("DOWN", self.move(0, 1)))
        if self.position_empty_space.x > 0:
            moves.append(("LEFT", self.move(-1, 0)))
        if self.position_empty_space.x < len(self.board[0]) - 1:
            moves.append(("RIGHT", self.move(1, 0)))
        return moves

    def to_tuple(self):
        return tuple(tuple(row) for row in self.board)

def heuristic(board):
    """Calcula la distancia de Manhattan como heurística"""
    goal_positions = {Game.GOAL[y][x]: (x, y) for y in range(3) for x
in range(3) if Game.GOAL[y][x] is not None}
    total_distance = 0
    for y in range(3):
        for x in range(3):
            value = board[y][x]
            if value is not None:
                goal_x, goal_y = goal_positions[value]
                total_distance += abs(x - goal_x) + abs(y - goal_y)
    return total_distance

def astar(game):
    open_set = []
    visited = set()

    initial_h = heuristic(game.board)
    start_node = Node(game.board, None, 0, initial_h)
    heapq.heappush(open_set, (start_node.cost + start_node.heuristic,
start_node, []))

    while open_set:
        _, current_node, path = heapq.heappop(open_set)

        current_game = Game(current_node.stage)
        if current_game.is_game_win():
            return path

        board_tuple = current_game.to_tuple()
```

```
        if board_tuple in visited:
            continue
        visited.add(board_tuple)

        for move, next_game in current_game.next_allowed_moves():
            new_g = current_node.cost + 1  # Costo acumulado
            new_h = heuristic(next_game.board)
            new_node = Node(next_game.board, move, new_g, new_h)
            heapq.heappush(open_set, (new_node.cost +
new_node.heuristic, new_node, path + [move]))

    return None

initial_board = [
    [None, 8, 7],
    [5, 4, 6],
    [3, 2, 1]
]

game = Game(initial_board)
print("Tablero inicial:")
game.show()

first_time = datetime.datetime.now()

solution_moves = astar(game)

later_time = datetime.datetime.now()

if solution_moves is not None:
    print("\nSecuencia de movimientos para alcanzar el GOAL:")
    for move in solution_moves:
        print(move)
    print("La cantidad de movimientos sugeridos usando A* es:",
len(solution_moves))
else:
    print("No se encontró solución.")

print(f"El tiempo de ejecución usando A*: {later_time - first_time}")
```

```
Tablero inicial:
```




```
Secuencia de movimientos para alcanzar el GOAL:
RIGHT
DOWN
RIGHT
DOWN
LEFT
LEFT
UP
UP
RIGHT
RIGHT
DOWN
DOWN
LEFT
UP
UP
RIGHT
DOWN
DOWN
LEFT
LEFT
UP
RIGHT
DOWN
LEFT
```

```
UP
UP
RIGHT
RIGHT
DOWN
DOWN
La cantidad de movimientos sugeridos usando A* es: 30
El tiempo de ejecución usando A*: 0:00:01.622414
```