A Star (A*)

In [ ]:
```
import pygame
import sys
import heapq
import time

pygame.init()

# Pengaturan layar
width, height = 900, 900
screen = pygame.display.set_mode((width, height))
pygame.display.set_caption('8-Puzzle with A* Search')

# Warna
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)

class PuzzleState:
    def __init__(self, board, empty_pos, moves=0, prev_state=None):
        self.board = board
        self.empty_pos = empty_pos
        self.moves = moves
        self.prev_state = prev_state

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

    def heuristic(self):
        manhattan_distance = 0
        for i in range(3):
            for j in range(3):
                value = self.board[i][j]
                if value != 0:
                    target_x, target_y = divmod(value - 1, 3)
                    manhattan_distance += abs(target_x - i) + abs(target_
        return manhattan_distance + self.moves  # Menambahkan jumlah lang

    def is_goal(self):
        goal = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]
        return self.board == goal

    def get_neighbors(self):
        x, y = self.empty_pos
        directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
        neighbors = []

        for dx, dy in directions:
            new_x, new_y = x + dx, y + dy
            if 0 <= new_x < 3 and 0 <= new_y < 3:
                new_board = [row[:] for row in self.board]
                new_board[x][y], new_board[new_x][new_y] = new_board[new_
                neighbors.append(PuzzleState(new_board, (new_x, new_y), s

        return neighbors

def astar(start_state):
    frontier = []
    heapq.heappush(frontier, start_state)
    visited = set()
    visited.add(tuple(map(tuple, start_state.board)))

    while frontier:
        current_state = heapq.heappop(frontier)

        if current_state.is_goal():
            return current_state

        for neighbor in current_state.get_neighbors():
            state_tuple = tuple(map(tuple, neighbor.board))
            if state_tuple not in visited:
                visited.add(state_tuple)
                heapq.heappush(frontier, neighbor)

    return None

def get_solution_path(final_state):
    path = []
    current_state = final_state
    while current_state:
        path.append(current_state)
        current_state = current_state.prev_state
    return path[::-1]

def draw_board(board):
    screen.fill(WHITE)
    tile_size = width // 3  # Ukuran tile untuk ukuran 900x900
    font_size = tile_size // 2  # Ukuran font sesuai ukuran tile
    font = pygame.font.Font(None, font_size)

    for i in range(3):
        for j in range(3):
            value = board[i][j]
            if value != 0:
                pygame.draw.rect(screen, BLACK, (j * tile_size, i * tile_
                text = font.render(str(value), True, WHITE)
                text_rect = text.get_rect(center=(j * tile_size + tile_si
                screen.blit(text, text_rect)

def print_board(board):
    for row in board:
        print(row)
```

```python
def game_loop():
    # Kondisi awal puzzle
    initial_board = [[3, 2, 1], [4, 0, 5], [6, 8, 7]]
    empty_pos = (1, 1)
    puzzle_state = PuzzleState(initial_board, empty_pos)

    print("Mencari solusi dengan A* Search...")
    solution = astar(puzzle_state)

    if solution:
        print("Solusi ditemukan!")

        # Ambil jalur solusi
        solution_path = get_solution_path(solution)
        total_steps = len(solution_path) - 1

        print(f"Solusi ditemukan dalam {total_steps} langkah:")

        # Tampilkan setiap langkah dari solusi
        for state in solution_path:
            print_board(state.board)
            print()  # Print kosong untuk jarak antar langkah
            draw_board(state.board)
            pygame.display.flip()
            time.sleep(1)

    running = True
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
        pygame.display.update()

    pygame.quit()
    sys.exit()

# Menjalankan game
game_loop()
```

```
pygame 2.6.0 (SDL 2.28.4, Python 3.11.3)
Hello from the pygame community. https://www.pygame.org/contribute.html
Mencari solusi dengan A* Search...
Solusi ditemukan!
Solusi ditemukan dalam 24 langkah:
[3, 2, 1]
[4, 0, 5]
[6, 8, 7]

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

[3, 2, 1]
[4, 8, 5]
```

```
[6, 7, 0]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Greedy Best-first Search Algorithm

In [ ]:
```python
import pygame
import sys
import heapq
import time

pygame.init()

# Pengaturan layar
width, height = 900, 900
screen = pygame.display.set_mode((width, height))
pygame.display.set_caption('8-Puzzle with Greedy Best-First Search')
```

```
# Warna
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)

class PuzzleState:
    def __init__(self, board, empty_pos, moves=0, prev_state=None):
        self.board = board
        self.empty_pos = empty_pos
        self.moves = moves
        self.prev_state = prev_state

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

    def heuristic(self):
        # Heuristik menggunakan Manhattan Distance (GBFS hanya menggunaka
        manhattan_distance = 0
        for i in range(3):
            for j in range(3):
                value = self.board[i][j]
                if value != 0:
                    target_x, target_y = divmod(value - 1, 3)
                    manhattan_distance += abs(target_x - i) + abs(target_
        return manhattan_distance  # Hanya heuristik, tanpa menambahkan s

    def is_goal(self):
        goal = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]
        return self.board == goal

    def get_neighbors(self):
        x, y = self.empty_pos
        directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
        neighbors = []

        for dx, dy in directions:
            new_x, new_y = x + dx, y + dy
            if 0 <= new_x < 3 and 0 <= new_y < 3:
                new_board = [row[:] for row in self.board]
                new_board[x][y], new_board[new_x][new_y] = new_board[new_
                neighbors.append(PuzzleState(new_board, (new_x, new_y), s

        return neighbors

def greedy_best_first(start_state):
    frontier = []
    heapq.heappush(frontier, start_state)
    visited = set()
    visited.add(tuple(map(tuple, start_state.board)))

    while frontier:
        current_state = heapq.heappop(frontier)
        if current_state.is_goal():
            return current_state

        for neighbor in current_state.get_neighbors():
            state_tuple = tuple(map(tuple, neighbor.board))
            if state_tuple not in visited:
                visited.add(state_tuple)
                heapq.heappush(frontier, neighbor)

    return None

def get_solution_path(final_state):
    path = []
    current_state = final_state
    while current_state:
        path.append(current_state)
        current_state = current_state.prev_state
    return path[::-1]

def draw_board(board):
    screen.fill(WHITE)
    tile_size = width // 3  # Ukuran tile untuk ukuran 900x900
    font_size = tile_size // 2  # Ukuran font sesuai ukuran tile
    font = pygame.font.Font(None, font_size)

    for i in range(3):
        for j in range(3):
            value = board[i][j]
            if value != 0:
                pygame.draw.rect(screen, BLACK, (j * tile_size, i * tile_
                text = font.render(str(value), True, WHITE)
                text_rect = text.get_rect(center=(j * tile_size + tile_si
                screen.blit(text, text_rect)

def print_board(board):
    for row in board:
        print(row)

def game_loop():
    # Kondisi awal puzzle
    initial_board = [[3, 2, 1], [4, 0, 5], [6, 8, 7]]
    empty_pos = (1, 1)
    puzzle_state = PuzzleState(initial_board, empty_pos)

    print("Mencari solusi dengan Greedy Best-First Search...")
    solution = greedy_best_first(puzzle_state)

    if solution:
        print("Solusi ditemukan!")

        # Ambil jalur solusi
        solution_path = get_solution_path(solution)
        total_steps = len(solution_path) - 1
        print(f"Solusi ditemukan dalam {total_steps} langkah:")

        # Tampilkan setiap langkah dari solusi
        for state in solution_path:
            print_board(state.board)
            print()  # Print kosong untuk jarak antar langkah
            draw_board(state.board)
            pygame.display.flip()
            time.sleep(1)

    running = True
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
        pygame.display.update()

    pygame.quit()
    sys.exit()

# Menjalankan game
game_loop()
```

```
pygame 2.6.0 (SDL 2.28.4, Python 3.11.3)
Hello from the pygame community. https://www.pygame.org/contribute.html
Mencari solusi dengan Greedy Best-First Search...
Solusi ditemukan!
Solusi ditemukan dalam 72 langkah:
[3, 2, 1]
[4, 0, 5]
[6, 8, 7]

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

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

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

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

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

[4, 3, 2]
```

```
        [6, 5, 1]
        [0, 8, 7]

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

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

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

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

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

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

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

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

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

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

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

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

        [3, 2, 5]
        [4, 7, 0]
```

```
[8, 1, 6]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[1, 5, 3]
```

```
[0, 2, 4]
[7, 8, 6]

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

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

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

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

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

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

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

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

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

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

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

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

[1, 2, 3]
[4, 5, 6]
```

```
[7, 8, 0]
```