BAI501
F23

الطلاب المشاركين:

aya_187934 – C1 آية فتال يبرودي

lubna_175170 – C1 لبنى بكداش

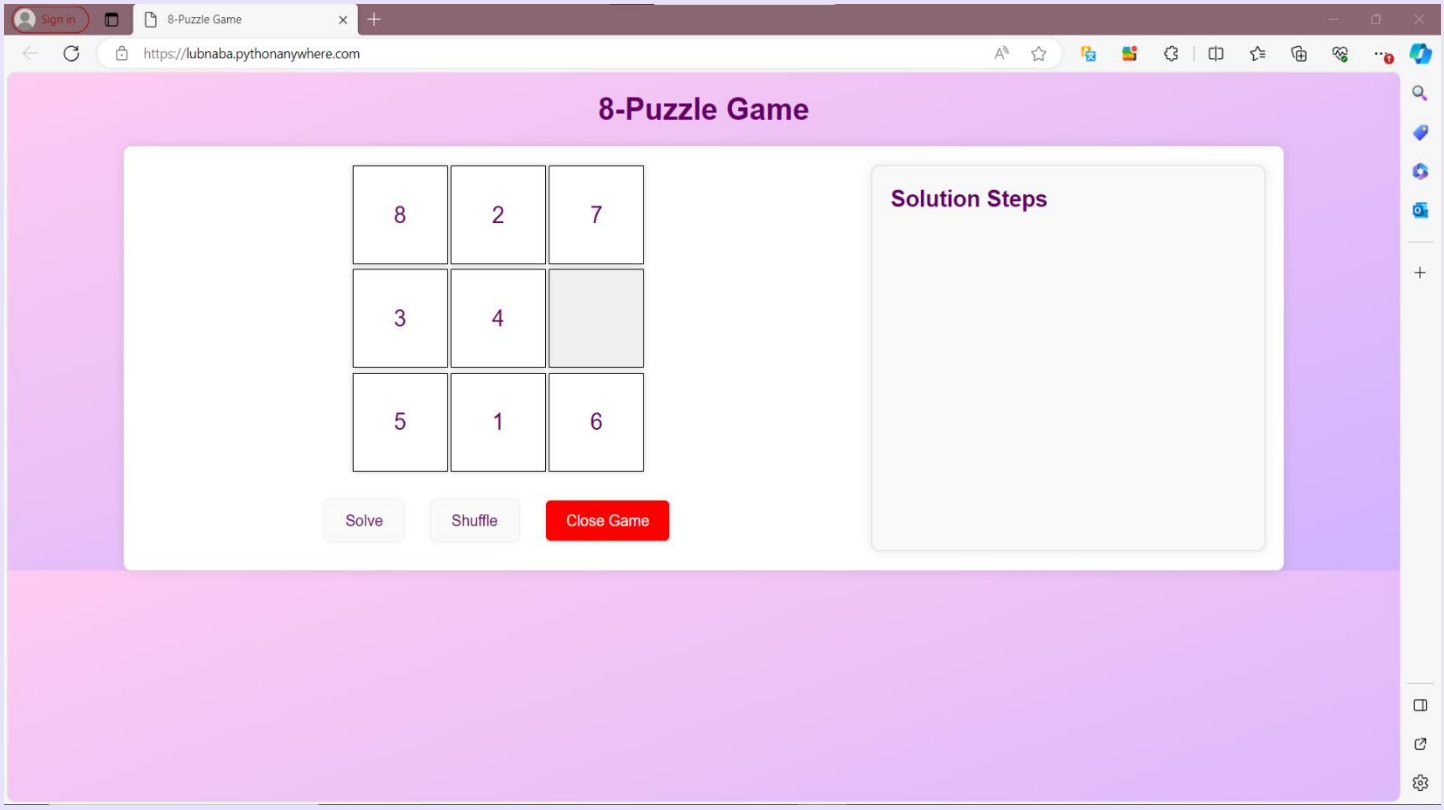
abd_arrhman_230288-C1 عبد الرحمن أبو هايلة

nagham_191378– C1 نغم الشامي

➢ رابط اللعبة: 8-Puzzle Game (lubnaba.pythonanywhere.com)

➢ واجهة اللعبة:

❖ تمت برمجة اللعبة باستخدام لغة Python وباستخدام خوارزمية *A.



❖ خوارزمية البحث*A

خوارزمية البحث *A هي خوارزمية تستخدم في الذكاء الاصطناعي والمجالات المتعلقة بعلوم الكمبيوتر، مثل حل الألغاز، وتخطيط المسارات، وألعاب الفيديو، وغيرها.

تجمع خوارزمية *A بين خوارزميتين أساسيتين: البحث بالعرض الأول وخوارزمية الاستدلال، مما يجعلها فعالة في العثور على أقصر مسار في رسم بياني.

➢ أكواد الواجهة:

❖ أكواد html:

index.html

C: > Users > ASUS > Desktop > 8_puzzle > templates > index.html > ...

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>8-Puzzle Game</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>

<body>
    <h1 class="title">8-Puzzle Game</h1>
    <div class="container">
        <div class="game-area">
            <div class="board" id="board">
                <!-- Tiles will be dynamically generated here -->
            </div>
            <div class="controls">
                <button onclick="solvePuzzle()" class="solve-button">Solve</button>
                <button onclick="shuffleBoard()" class="shuffle-button">Shuffle</button>
                <button onclick="closeGame()" class="close-button">Close Game</button>

            </div>
        </div>
        <div class="solution-area">
            <h2>Solution Steps</h2>
            <div id="solution"></div>
        </div>
    </div>
    <script src="{{ url_for('static', filename='main.js') }}"></script>
</body>

</html>
```

❖ أكواد JS:

main.js

static > main.js > document.addEventListener("DOMContentLoaded") callback > moveTile

```
document.addEventListener("DOMContentLoaded", () => {
    let board = [
        [1, 2, 3],
        [4, 0, 5],
        [7, 8, 6]
    ];

    const boardElement = document.getElementById("board");

    function renderBoard(board) {
        boardElement.innerHTML = "";
        board.forEach((row, i) => {
            row.forEach((tile, j) => {
                const tileElement = document.createElement("div");
                tileElement.className = "tile";
                tileElement.innerText = tile === 0 ? "" : tile;
                tileElement.classList.toggle("empty", tile === 0);
                tileElement.addEventListener("click", () => moveTile(i, j));
                boardElement.appendChild(tileElement);
            });
        });
    }

    function findEmptyTile() {
        for (let i = 0; i < 3; i++) {
            for (let j = 0; j < 3; j++) {
                if (board[i][j] === 0) return [i, j];
            }
        }
    }

    function moveTile(i, j) {
        const [emptyI, emptyJ] = findEmptyTile();
        if ((Math.abs(emptyI - i) === 1 && emptyJ === j) || (Math.abs(emptyJ - j) === 1 && emptyI === i)) {
            [board[emptyI][emptyJ], board[i][j]] = [board[i][j], board[emptyI][emptyJ]];
            renderBoard(board);
        }
```

main.js

static > main.js > document.addEventListener("DOMContentLoaded") callback > moveTile

```
document.addEventListener("DOMContentLoaded", () => {

    window.solvePuzzle = function () {
        fetch('/solve', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({ board })
        })
            .then(response => response.json())
            .then(data => {
                const solution = data.solution;
                const error = data.error;
                const solutionElement = document.getElementById("solution");
                solutionElement.innerHTML = "";
                if (error) {
                    alert(error);
                } else if (solution) {
                    solution.forEach((step, index) => {
                        const stepElement = document.createElement("div");
                        stepElement.innerHTML = `<h3>Step ${index + 1}</h3>`;
                        step.forEach(row => {
                            const rowElement = document.createElement("div");
                            rowElement.innerText = row.join(" ");
                            stepElement.appendChild(rowElement);
                        });
                        solutionElement.appendChild(stepElement);
                    });
                } else {
                    alert("No solution found!");
                }
            });
    };

    window.shuffleBoard = function () {
        fetch('/shuffle')
            .then(response => response.json())
            .then(data => {
                board = data.board;
                renderBoard(board);
            });
    };

    renderBoard(board);
});
function closeGame() {
    if (confirm("Are you sure you want to close the game?")) {
        window.close();
    }
}

```

❖ أكود app.py:

هذا الكود كتب بلغة البايثون واستخدمنا من خلاله تطبيق Flask لحل لعبة القطع الثمانية باستخدام خوارزمية A*.

app.py 1 ×

app.py > solve

```python
from flask import Flask, render_template, request, jsonify
from puzzle_solver import a_star_search, get_solution_path, generate_random_board, is_solvable

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/solve', methods=['POST'])
def solve():
    board = request.json['board']
    if not is_solvable(board):
        return jsonify({"solution": None, "error": "Board is not solvable"})
    solution_state = a_star_search(board)
    if solution_state:
        path = get_solution_path(solution_state)
        return jsonify({"solution": path})
    return jsonify({"solution": None, "error": "No solution found"})

@app.route('/shuffle', methods=['GET'])
def shuffle():
    board = generate_random_board()
    return jsonify({"board": board})

if __name__ == '__main__':
    app.run(debug=True)

```

في هذه الأكواد قمنا ب:

- استدعاء المكتبات والوظائف ثم إنشاء تطبيق Flask ثم أكواد حل اللغز ابتداء من مسار تلقي طلبات حل اللغز الى دالة المعالجة.
- الحصول على حالة اللغز الحالية والتحقق فيما إذا كان اللغز قابل للحل.
- حل اللغز باستخدام خوارزمية A* والحصول على تسلسل حل اللغز.
- توليد لوحة عشوائية وتشغيل التطبيق.

❖ أكود Puzzle_Solver.py:

هذه الأكود تمثل حلًا للعبة القطع الثمانية باستخدام خوارزمية البحث A* .

puzzle_solver.py

```python
import heapq
import random

class PuzzleState:
    def __init__(self, board, moves=0, previous=None):
        self.board = board
        self.moves = moves
        self.previous = previous
        self.heuristic = self.calculate_heuristic()

    def calculate_heuristic(self):
        """ Calculate the Manhattan distance heuristic """
        goal = {1: (0, 0), 2: (0, 1), 3: (0, 2), 4: (1, 0), 5: (1, 1), 6: (1, 2), 7: (2, 0), 8: (2, 1)}
        distance = 0
        for i in range(3):
            for j in range(3):
                value = self.board[i][j]
                if value != 0:
                    target_x, target_y = goal[value]
                    distance += abs(i - target_x) + abs(j - target_y)
        return distance
```

1

في هذه الأكواد قمنا ب:

- استيراد المكتبات.
- تعريف صف class لحل اللغز.
- حساب قيمة الاستدلال: من خلال calculate heuristic تحسب المسافة Manhattan distance، وهي مجموع المسافات بين كل قطعة وموقعها الهدف.

```
22
23      def get_neighbors(self):
24          """ Generate neighbors by sliding tiles """
25          def swap_and_create(new_board, i1, j1, i2, j2):
26              new_board[i1][j1], new_board[i2][j2] = new_board[i2][j2], new_board[i1][j1]
27              return new_board
28
29          neighbors = []
30          x, y = next((i, j) for i, row in enumerate(self.board) for j, val in enumerate(row) if val == 0)
31          directions = [(x-1, y), (x+1, y), (x, y-1), (x, y+1)]
32
33          for i, j in directions:
34              if 0 <= i < 3 and 0 <= j < 3:
35                  new_board = [row[:] for row in self.board]
36                  new_board = swap_and_create(new_board, x, y, i, j)
37                  neighbors.append(PuzzleState(new_board, self.moves + 1, self))
38
39          return neighbors
40
```

2

في هذا الكود قمنا ب:

- توليد الجيران: get neighbors تولد جميع الحالات الممكنة من خلال تحريك القطعة الفارغة (0) في جميع الاتجاهات الأربعة الممكنة.

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

3

في هذا الكود قمنا ب:

- التحقق من الحالة الهدف: is goal تتحقق إذا كانت الحالة الحالية هي الهدف.

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

4

في هذا الكود قمنا ب:

- المقارنة بين الحالات: lt تحدد كيفية مقارنة حالتين بناءً على قيمة الاستدلال وعدد الحركات، وهذا مهم لقائمة الأولويات.

puzzle_solver.py

puzzle_solver.py > PuzzleState > calculate_heuristic

```python
46
47  def a_star_search(start_board):
48      start_state = PuzzleState(start_board)
49      open_list = []
50      closed_list = set()
51
52      heapq.heappush(open_list, start_state)
53      while open_list:
54          current_state = heapq.heappop(open_list)
55
56          if current_state.is_goal():
57              return current_state
58
59          closed_list.add(tuple(map(tuple, current_state.board)))
60          for neighbor in current_state.get_neighbors():
61              if tuple(map(tuple, neighbor.board)) not in closed_list:
62                  heapq.heappush(open_list, neighbor)
63
64      return None
```

5

في هذا الكود قمنا ب:

- خوارزمية البحث *A: a_star_search تنفيذ خوارزمية البحث*A
- تبدأ بحالة البداية وتضيفها إلى قائمة الأولويات.
- تتكرر العملية حتى العثور على حالة الهدف أو نفاد الحالات الممكنة.
- تحقق من حالة الهدف، وإذا كانت غير موجودة في القائمة المغلقة (visited list)، تضاف إلى قائمة الأولويات.

```python
def get_solution_path(state):
    path = []
    while state:
        path.append(state.board)
        state = state.previous
    return path[::-1]

def is_solvable(board):
    flat_board = [num for row in board for num in row if num != 0]
    inversions = 0
    for i in range(len(flat_board)):
        for j in range(i + 1, len(flat_board)):
            if flat_board[i] > flat_board[j]:
                inversions += 1
    return inversions % 2 == 0
```

6

في هذا الكود قمنا ب:

- تتبع مسار الحل :get_solution_path تُرجع المسار من الحالة الأولية إلى الحالة الهدف من خلال تتبع الحالات السابقة.
- التحقق من قابلية الحل: is solvable تتحقق من قابلية حل اللغز عن طريق حساب عدد الانعكاسات .

```python
def generate_random_board():
    while True:
        board = [1, 2, 3, 4, 5, 6, 7, 8, 0]
        random.shuffle(board)
        board = [board[i:i + 3] for i in range(0, 9, 3)]
        if is_solvable(board):
            return board
```

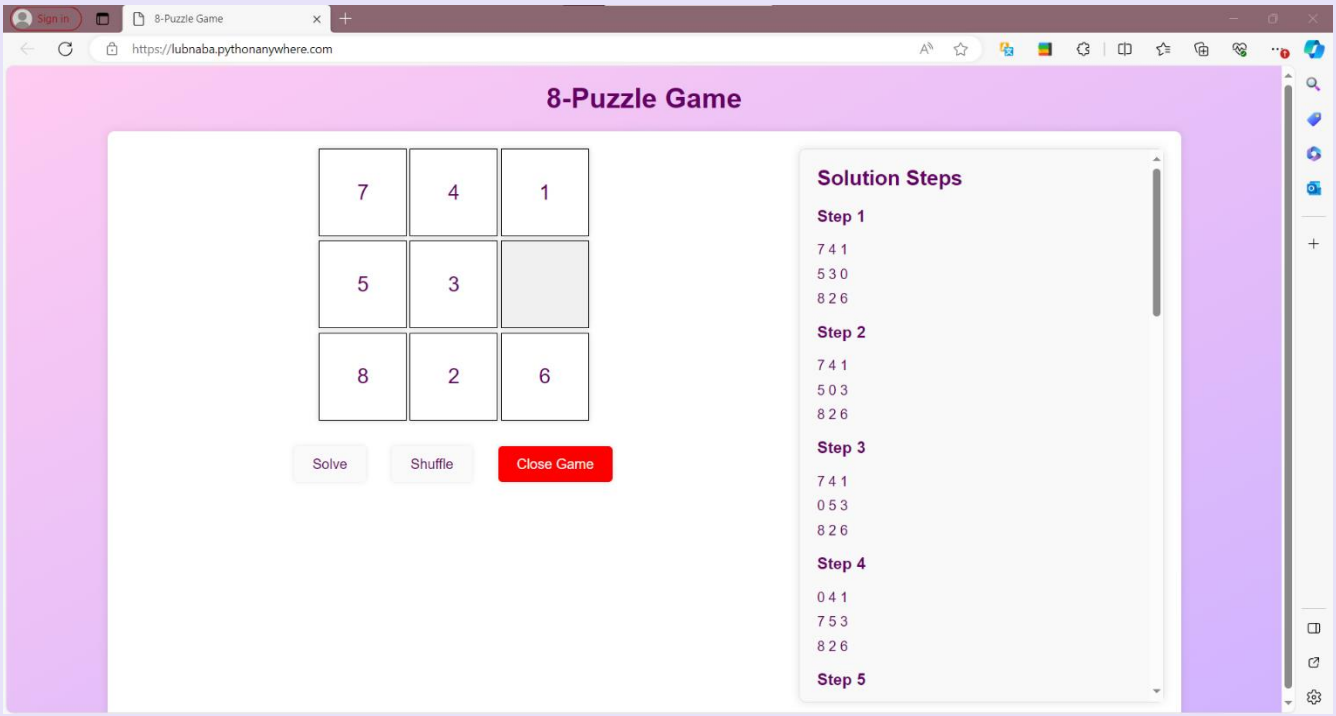
7

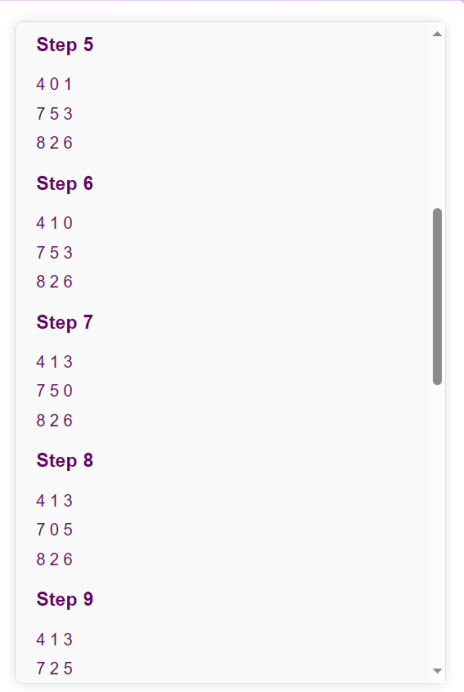
في هذا الكود قمنا ب:

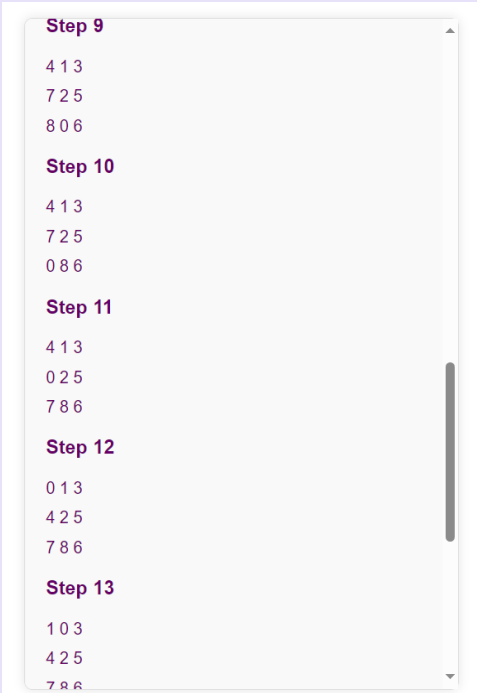
- توليد لوحة عشوائية: generate_random_board تولد لوحة عشوائية جديدة وتتحقق من قابليتها للحل.

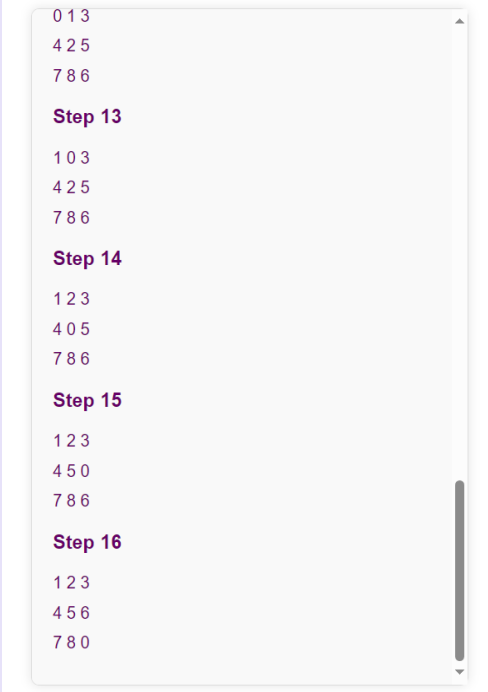
➢ تسلسل الحل:

❖ عند الضغط على زر Solve تقوم اللعبة بإعطاء تسلسل خطوات الحل للاعب حتى يقوم بالوصول لحل هذه اللعبة.







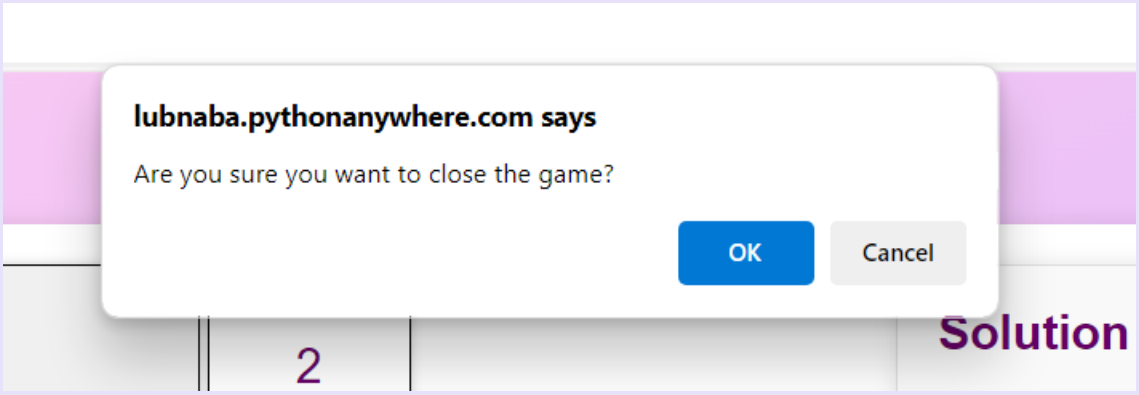


➢ زر Shuffle:

- ❖ عند الضغط على زر Shuffle تقوم اللعبة بخلط قطعها الثمانية وتغيير أماكنها وبالتالي توليد لعبة جديدة للاعب وتقوم بتوفير حل جديد بناءً على أماكن القطع الجديدة.

➢ زر Close Game:

- ❖ عند الضغط على زر Close Game يتم سؤال المستخدم فيما اذا كان يود الخروج من اللعبة وعند الحصول على التأكيد يتم إغلاق صفحة الويب والخروج من اللعبة.



➢ تحريك القطع:

- ❖ يستطيع اللاعب تحريك القطع للعب بهذه اللعبة ومحاولة القيام بحلها وعند عدم قدرته على حلها يمكنه الضغوط على زر Solve للحصول على الحل.

➢ الأكواد للنسخ:

❖ App.py

```
from flask import Flask, render_template, request, jsonify
from puzzle_solver import a_star_search, get_solution_path,
generate_random_board, is_solvable

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/solve', methods=['POST'])
def solve():
    board = request.json['board']
    if not is_solvable(board):
        return jsonify({"solution": None, "error": "Board is not
solvable"})
    solution_state = a_star_search(board)
    if solution_state:
        path = get_solution_path(solution_state)
        return jsonify({"solution": path})
    return jsonify({"solution": None, "error": "No solution found"})

@app.route('/shuffle', methods=['GET'])
def shuffle():
    board = generate_random_board()
    return jsonify({"board": board})

if __name__ == '__main__':
    app.run(debug=True)
```

❖ Puzzle_Solver

```
import heapq
import random

class PuzzleState:
    def __init__(self, board, moves=0, previous=None):
        self.board = board
        self.moves = moves
        self.previous = previous
        self.heuristic = self.calculate_heuristic()

    def calculate_heuristic(self):
        """ Calculate the Manhattan distance heuristic """
        goal = {1: (0, 0), 2: (0, 1), 3: (0, 2), 4: (1, 0), 5: (1, 1), 6:
(1, 2), 7: (2, 0), 8: (2, 1)}
        distance = 0
        for i in range(3):
            for j in range(3):
                value = self.board[i][j]
                if value != 0:
                    target_x, target_y = goal[value]
                    distance += abs(i - target_x) + abs(j - target_y)
        return distance

    def get_neighbors(self):
        """ Generate neighbors by sliding tiles """
        def swap_and_create(new_board, i1, j1, i2, j2):
            new_board[i1][j1], new_board[i2][j2] = new_board[i2][j2],
new_board[i1][j1]
            return new_board

        neighbors = []
        x, y = next((i, j) for i, row in enumerate(self.board) for j, val
in enumerate(row) if val == 0)
        directions = [(x-1, y), (x+1, y), (x, y-1), (x, y+1)]

        for i, j in directions:
            if 0 <= i < 3 and 0 <= j < 3:
                new_board = [row[:] for row in self.board]
                new_board = swap_and_create(new_board, x, y, i, j)
                neighbors.append(PuzzleState(new_board, self.moves + 1,
self))
```

```
        return neighbors

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

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

def a_star_search(start_board):
    start_state = PuzzleState(start_board)
    open_list = []
    closed_list = set()

    heapq.heappush(open_list, start_state)
    while open_list:
        current_state = heapq.heappop(open_list)

        if current_state.is_goal():
            return current_state

        closed_list.add(tuple(map(tuple, current_state.board)))
        for neighbor in current_state.get_neighbors():
            if tuple(map(tuple, neighbor.board)) not in closed_list:
                heapq.heappush(open_list, neighbor)

    return None

def get_solution_path(state):
    path = []
    while state:
        path.append(state.board)
        state = state.previous
    return path[::-1]

def is_solvable(board):
    flat_board = [num for row in board for num in row if num != 0]
    inversions = 0
    for i in range(len(flat_board)):
        for j in range(i + 1, len(flat_board)):
            if flat_board[i] > flat_board[j]:
                inversions += 1
    return inversions % 2 == 0
```

```
def generate_random_board():
    while True:
        board = [1, 2, 3, 4, 5, 6, 7, 8, 0]
        random.shuffle(board)
        board = [board[i:i + 3] for i in range(0, 9, 3)]
        if is_solvable(board):
            return board
```

==========

➢ الخاتمة

وفي الختام نود أن نتوجه بكامل الشكر والتقدير والاحترام للدكتور باسل الخطيب على جهوده المبذولة ودوره الرئيسي في مسيرتنا التعليمية ونؤكد أن جهودك ستثمر معنا

وسنحصد ثمارها ونحن نذكرك بكل خير.

شكراً على جهودكم مع تمنياتنا لكم بالصحة والعافية الدائمة.