جامعة الأزهر
AL-AZHAR UNIVERSITY
LOGO.ADAM96.COM

AI PROJECT

Under supervision: ENG. MOHAMMED ELREFAEI

OUR TEAM

AL-Hassan Amir

Set nu: 20192020
Uni code: 404033

Abdelalieem Ahmed

Set nu: 20192020
Uni code: 404033

Omar mokhtar

Set nu: 20192020
Uni code: 404033

Abdullah muhamed

Set nu: 20192020
Uni code: 404033

Muhamed mustafa

Set nu: 20192020
Uni code: 404033

The problem

8-puzzle problem is a puzzle invented and popularized by Noyes Palmer Chapman in the 1870s. It is played on a 3-by-3 grid with 8 square blocks labeled 1 through 8 and a blank square. It is required to rearrange the blocks in the order required using the empty space. you can slide four adjacent (left, right, above and below) blocks into the empty space.

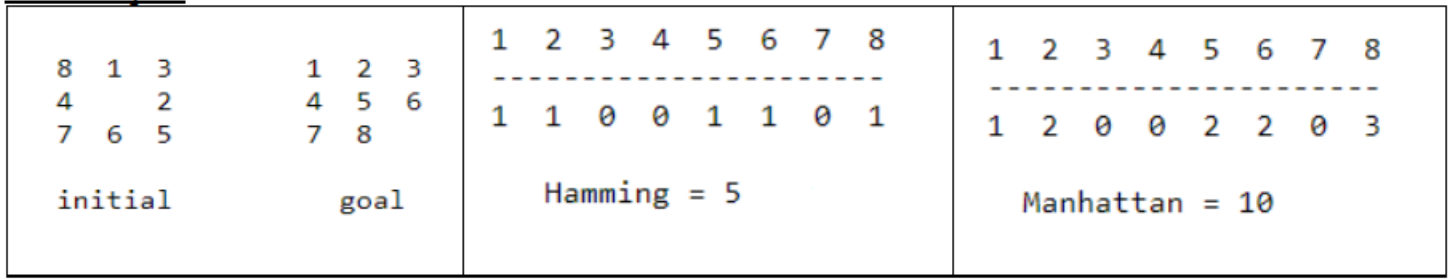
Search strategies to be applied:

- **Depth-First**
- **Breadth-First**
- **Best-First**

Note: In Best-First; the success of this approach depends on the choice of priority function for a state. consider two priority functions:

- *Hamming priority function*. The number of tiles in the wrong position. A state with a small number of blocks in the wrong position is close to the goal state, so it will be preferred.
- *Manhattan priority function*. The sum of the distances (sum of the vertical and horizontal distance) from the blocks to their goal positions, so it will be preferred.

For example:



The objective: build a problem solver for 8-puzzle game.

1) Implement the solution for the problem based on the above search techniques using the python.

Given the initial state:

1	8	2
	4	3
7	6	5

and the goal state:

1	2	3
4	5	6
7	8	

Make the program able to display:

-The solution.

-Number of states enqueued.

-Number of moves.:

2) Test the implementation for the instance (other initial state):

8	1	2
	4	3
7	6	5

and the goal state:

1	2	3
4	5	6
7	8	

Is that instance of 8-puzzle is solvable? Explain.

CODE

EXPLORER

OPEN EDITORS
- Welcome
- AI_COURSE_PROJECT

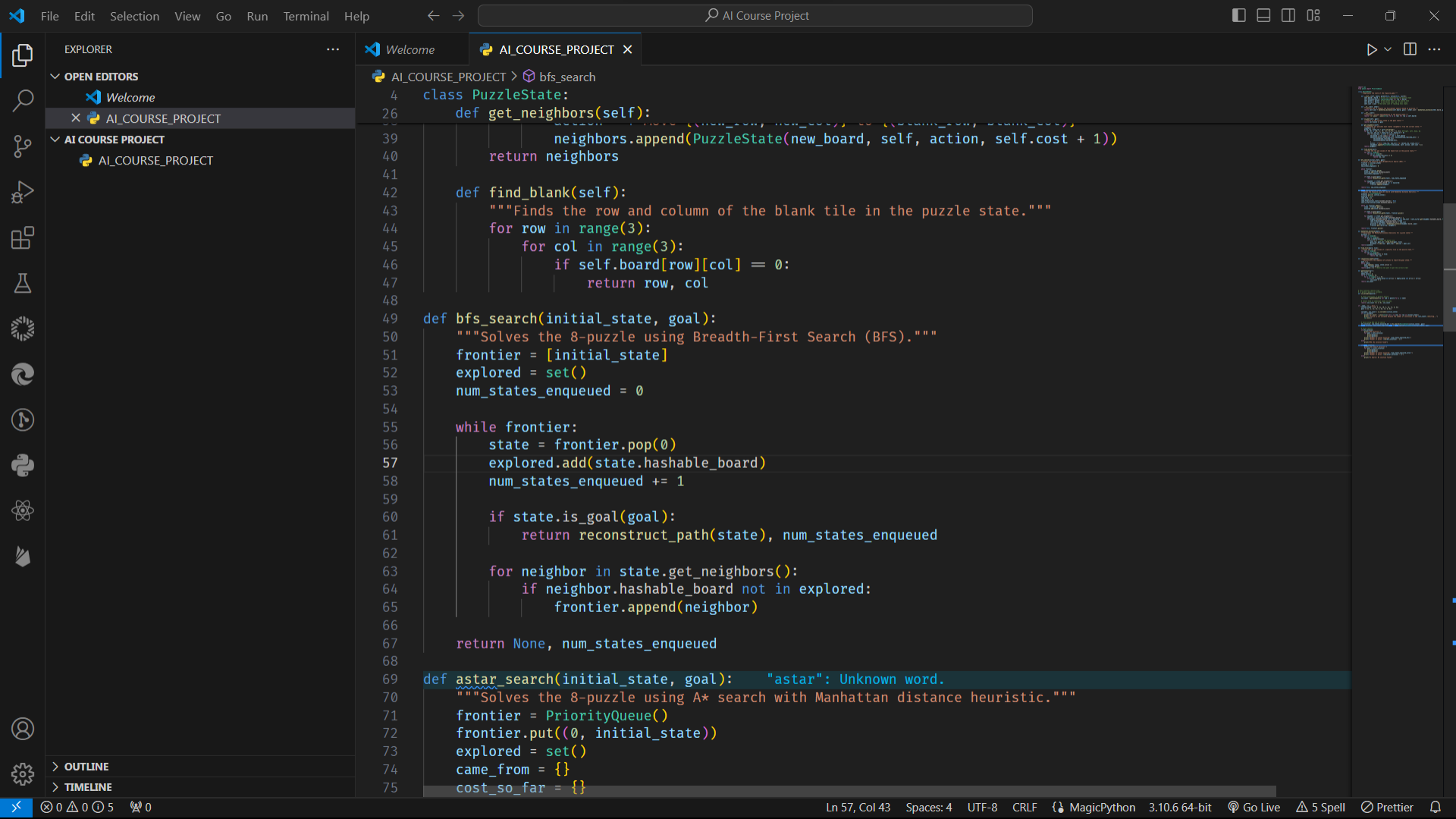
AI COURSE PROJECT
- AI_COURSE_PROJECT

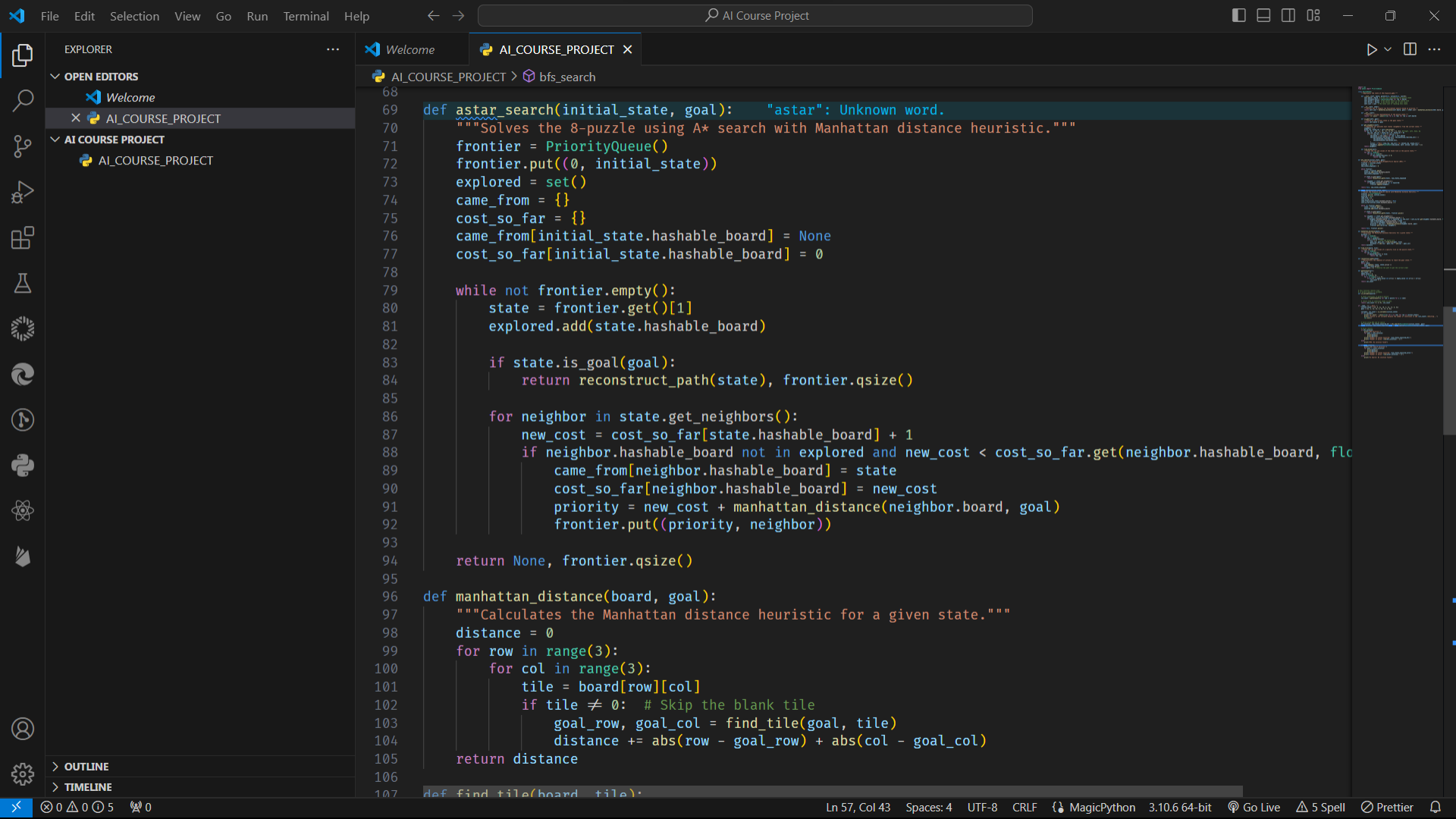
OUTLINE
TIMELINE

Welcome | AI_COURSE_PROJECT

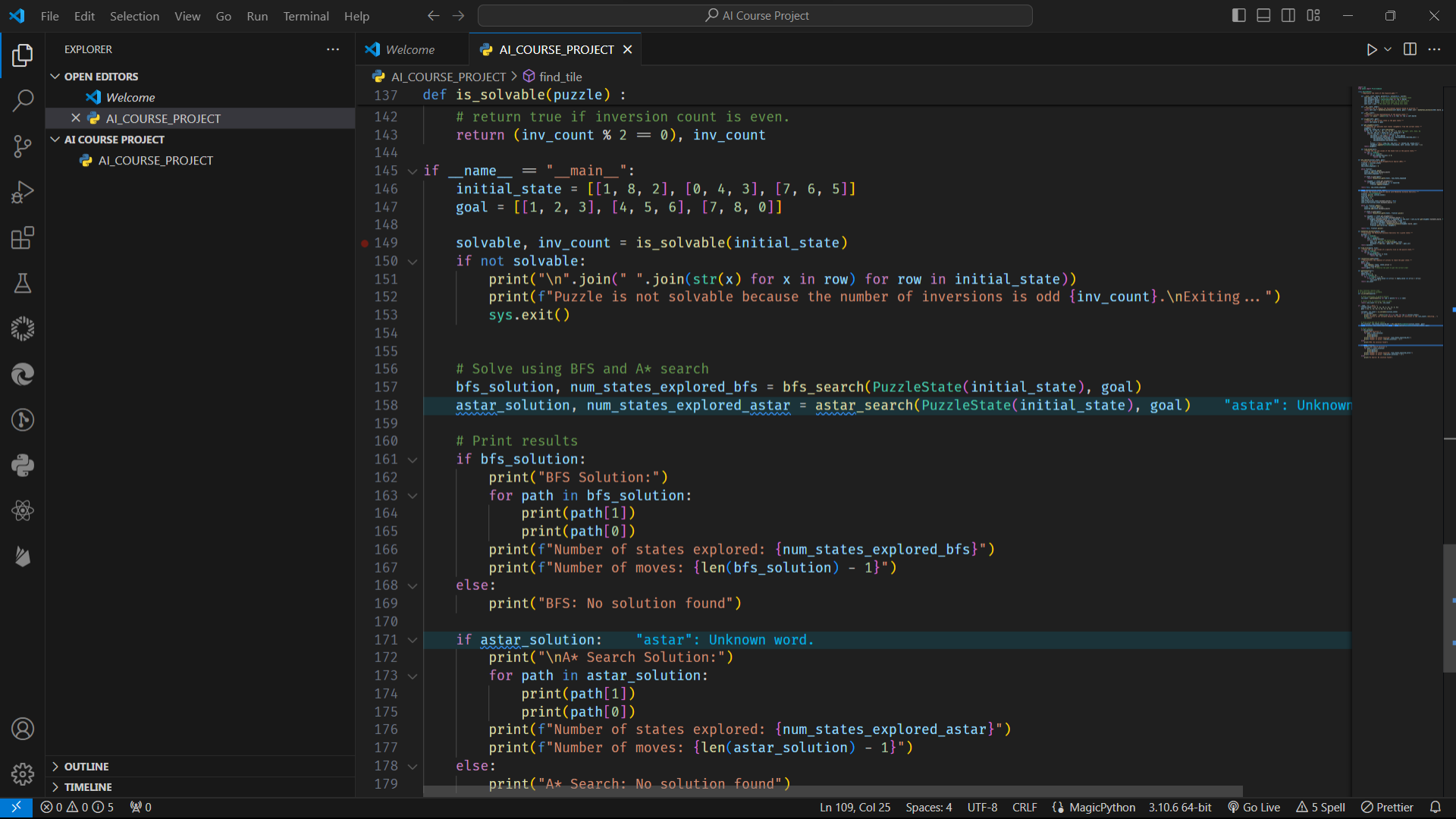
AI_COURSE_PROJECT > ...

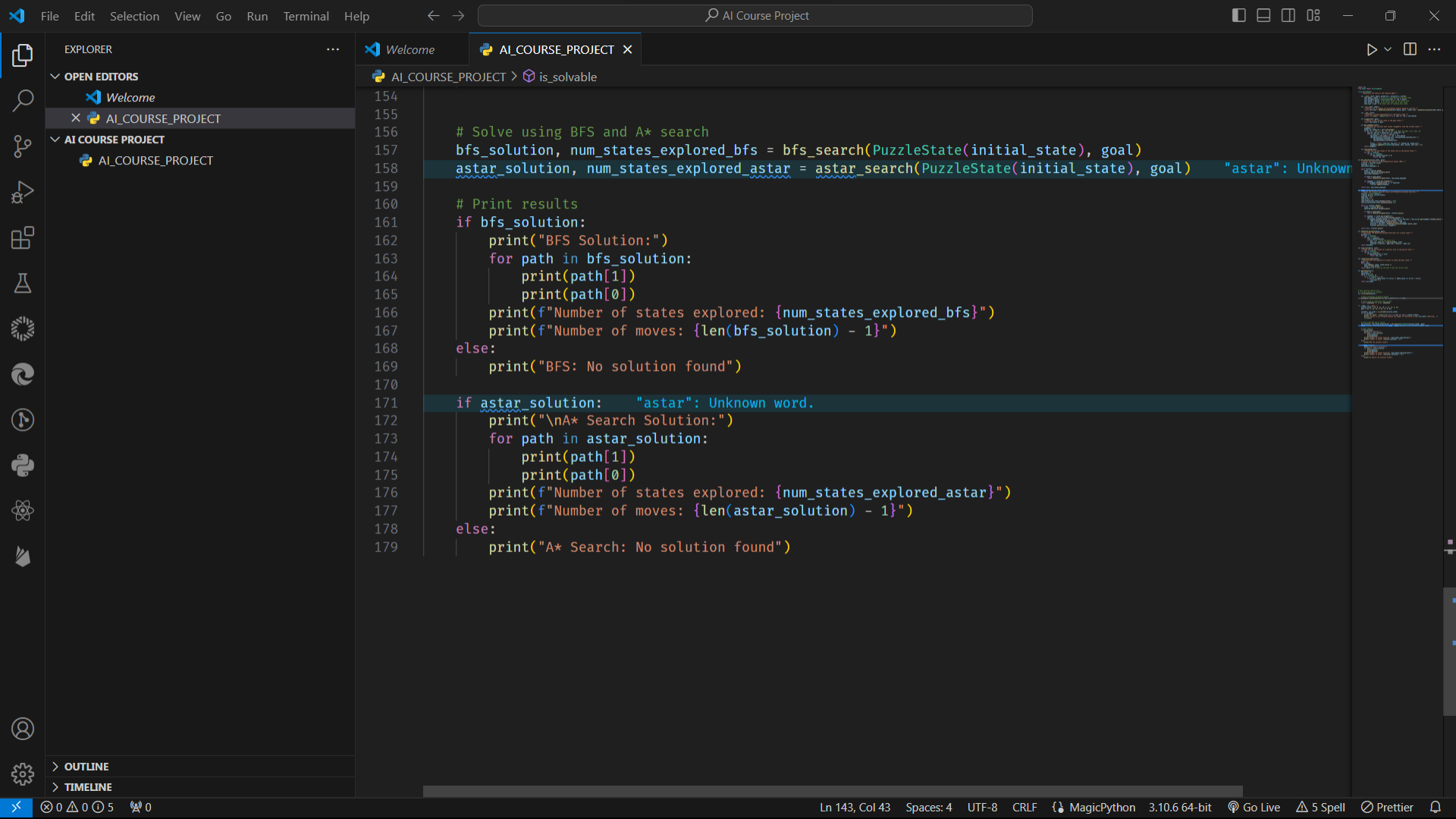
```python
import sys
from queue import PriorityQueue

class PuzzleState:
    """Represents the state of the 8-puzzle game."""

    def __init__(self, board, parent=None, action=None, cost=0):
        self.board = board  # A 3×3 list representing the puzzle state
        self.hashable_board = tuple(tuple(row) for row in board)
        self.parent = parent  # The parent state in the search tree
        self.action = action  # The action that led to this state
        self.cost = cost  # The total cost of reaching this state

    def __lt__(self, other):
        """Defines how to compare two PuzzleState objects based on priority."""
        return self.cost + manhattan_distance(self.board, goal) < other.cost + manhattan_distance(other.board, goa

    def __str__(self):
        """Returns a string representation of the puzzle state."""
        return "\n".join(" ".join(str(x) for x in row) for row in self.board)

    def is_goal(self, goal):
        """Checks if the current state is the goal state."""
        return self.board == goal

    def get_neighbors(self):
        """Generates all possible next states (neighbors) from the current state."""
        neighbors = []
        blank_row, blank_col = self.find_blank()
        for dr, dc in [(0, 1), (0, -1), (1, 0), (-1, 0)]: ## Right, Left, Down, Up
            new_row, new_col = blank_row + dr, blank_col + dc
            if 0 <= new_row < 3 and 0 <= new_col < 3:
                new_board = [row.copy() for row in self.board]
                new_board[blank_row][blank_col], new_board[new_row][new_col] = (
                    new_board[new_row][new_col],
                    new_board[blank_row][blank_col],
                )
                action = f"Move {(new_row, new_col)} to {(blank_row, blank_col)}"
                neighbors.append(PuzzleState(new_board, self, action, self.cost + 1))
        return neighbors
```


File Edit Selection View Go Run Terminal Help
AI Course Project
EXPLORER
OPEN EDITORS
Welcome
AI_COURSE_PROJECT
AI COURSE PROJECT
AI_COURSE_PROJECT
Welcome
AI_COURSE_PROJECT
AI_COURSE_PROJECT > bfs_search
4 class PuzzleState:
26 def get_neighbors(self):
39 neighbors.append(PuzzleState(new_board, self, action, self.cost + 1))
40 return neighbors
41
42 def find_blank(self):
43 """Finds the row and column of the blank tile in the puzzle state."""
44 for row in range(3):
45 for col in range(3):
46 if self.board[row][col] == 0:
47 return row, col
48
49 def bfs_search(initial_state, goal):
50 """Solves the 8-puzzle using Breadth-First Search (BFS)."""
51 frontier = [initial_state]
52 explored = set()
53 num_states_enqueued = 0
54
55 while frontier:
56 state = frontier.pop(0)
57 explored.add(state.hashable_board)
58 num_states_enqueued += 1
59
60 if state.is_goal(goal):
61 return reconstruct_path(state), num_states_enqueued
62
63 for neighbor in state.get_neighbors():
64 if neighbor.hashable_board not in explored:
65 frontier.append(neighbor)
66
67 return None, num_states_enqueued
68
69 def astar_search(initial_state, goal): "astar": Unknown word.
70 """Solves the 8-puzzle using A* search with Manhattan distance heuristic."""
71 frontier = PriorityQueue()
72 frontier.put((0, initial_state))
73 explored = set()
74 came_from = {}
75 cost_so_far = {}
OUTLINE
TIMELINE
Ln 57, Col 43 Spaces: 4 UTF-8 CRLF MagicPython 3.10.6 64-bit Go Live 5 Spell Prettier


File Edit Selection View Go Run Terminal Help
AI Course Project
EXPLORER
OPEN EDITORS
Welcome
AI_COURSE_PROJECT
AI COURSE PROJECT
AI_COURSE_PROJECT
Welcome
AI_COURSE_PROJECT
AI_COURSE_PROJECT > bfs_search
68
69 def astar_search(initial_state, goal):   "astar": Unknown word.
70 """Solves the 8-puzzle using A* search with Manhattan distance heuristic."""
71 frontier = PriorityQueue()
72 frontier.put((0, initial_state))
73 explored = set()
74 came_from = {}
75 cost_so_far = {}
76 came_from[initial_state.hashable_board] = None
77 cost_so_far[initial_state.hashable_board] = 0
78
79 while not frontier.empty():
80 state = frontier.get()[1]
81 explored.add(state.hashable_board)
82
83 if state.is_goal(goal):
84 return reconstruct_path(state), frontier.qsize()
85
86 for neighbor in state.get_neighbors():
87 new_cost = cost_so_far[state.hashable_board] + 1
88 if neighbor.hashable_board not in explored and new_cost < cost_so_far.get(neighbor.hashable_board, flo
89 came_from[neighbor.hashable_board] = state
90 cost_so_far[neighbor.hashable_board] = new_cost
91 priority = new_cost + manhattan_distance(neighbor.board, goal)
92 frontier.put((priority, neighbor))
93
94 return None, frontier.qsize()
95
96 def manhattan_distance(board, goal):
97 """Calculates the Manhattan distance heuristic for a given state."""
98 distance = 0
99 for row in range(3):
100 for col in range(3):
101 tile = board[row][col]
102 if tile != 0: # Skip the blank tile
103 goal_row, goal_col = find_tile(goal, tile)
104 distance += abs(row - goal_row) + abs(col - goal_col)
105 return distance
106
107 def find_tile(board, tile):
OUTLINE
TIMELINE
Ln 57, Col 43 Spaces: 4 UTF-8 CRLF MagicPython 3.10.6 64-bit Go Live 5 Spell Prettier

```python
def find_tile(board, tile):
    """Finds the row and column of a specific tile in the puzzle state."""
    for row in range(3):
        for col in range(3):
            if board[row][col] == tile:
                return row, col

def reconstruct_path(state):
    """Reconstructs the sequence of actions to reach the goal state."""
    path = []
    while state:
        path.append(( state, state.action ))
        state = state.parent
    return path[::-1]  # Reverse the path to get the correct order

def getInvCount(arr):
    inv_count = 0
    empty_value = 0
    for i in range(0, 9):
        for j in range(i + 1, 9):
            if arr[j] != empty_value and arr[i] != empty_value and arr[i] > arr[j]:
                inv_count += 1
    return inv_count




# This function returns true
# if given 8 puzzle is solvable.
def is_solvable(puzzle) :

    # Count inversions in given 8 puzzle
    inv_count = getInvCount([j for sub in puzzle for j in sub])

    # return true if inversion count is even.
    return (inv_count % 2 == 0), inv_count

if __name__ == "__main__":
```


File Edit Selection View Go Run Terminal Help
AI Course Project
EXPLORER
OPEN EDITORS
Welcome
AI_COURSE_PROJECT
AI COURSE PROJECT
AI_COURSE_PROJECT
Welcome
AI_COURSE_PROJECT
AI_COURSE_PROJECT > find_tile
def is_solvable(puzzle) :
return true if inversion count is even.
return (inv_count % 2 == 0), inv_count
if __name__ == "__main__":
initial_state = [[1, 8, 2], [0, 4, 3], [7, 6, 5]]
goal = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]
solvable, inv_count = is_solvable(initial_state)
if not solvable:
print("\n".join(" ".join(str(x) for x in row) for row in initial_state))
print(f"Puzzle is not solvable because the number of inversions is odd {inv_count}.\nExiting ... ")
sys.exit()
# Solve using BFS and A* search
bfs_solution, num_states_explored_bfs = bfs_search(PuzzleState(initial_state), goal)
astar_solution, num_states_explored_astar = astar_search(PuzzleState(initial_state), goal) "astar": Unknown
# Print results
if bfs_solution:
print("BFS Solution:")
for path in bfs_solution:
print(path[1])
print(path[0])
print(f"Number of states explored: {num_states_explored_bfs}")
print(f"Number of moves: {len(bfs_solution) - 1}")
else:
print("BFS: No solution found")
if astar_solution: "astar": Unknown word.
print("\nA* Search Solution:")
for path in astar_solution:
print(path[1])
print(path[0])
print(f"Number of states explored: {num_states_explored_astar}")
print(f"Number of moves: {len(astar_solution) - 1}")
else:
print("A* Search: No solution found")
OUTLINE
TIMELINE
Ln 109, Col 25 Spaces: 4 UTF-8 CRLF MagicPython 3.10.6 64-bit Go Live 5 Spell Prettier


File Edit Selection View Go Run Terminal Help
AI Course Project
EXPLORER
OPEN EDITORS
Welcome
AI_COURSE_PROJECT
AI COURSE PROJECT
AI_COURSE_PROJECT
Welcome
AI_COURSE_PROJECT
AI_COURSE_PROJECT > is_solvable
154
155
156 # Solve using BFS and A* search
157 bfs_solution, num_states_explored_bfs = bfs_search(PuzzleState(initial_state), goal)
158 astar_solution, num_states_explored_astar = astar_search(PuzzleState(initial_state), goal) "astar": Unknown
159
160 # Print results
161 if bfs_solution:
162 print("BFS Solution:")
163 for path in bfs_solution:
164 print(path[1])
165 print(path[0])
166 print(f"Number of states explored: {num_states_explored_bfs}")
167 print(f"Number of moves: {len(bfs_solution) - 1}")
168 else:
169 print("BFS: No solution found")
170
171 if astar_solution: "astar": Unknown word.
172 print("\nA* Search Solution:")
173 for path in astar_solution:
174 print(path[1])
175 print(path[0])
176 print(f"Number of states explored: {num_states_explored_astar}")
177 print(f"Number of moves: {len(astar_solution) - 1}")
178 else:
179 print("A* Search: No solution found")
OUTLINE
TIMELINE
Ln 143, Col 43 Spaces: 4 UTF-8 CRLF MagicPython 3.10.6 64-bit Go Live 5 Spell Prettier

RUN

File Edit Selection View Go Run Terminal Help

AI Course Project

EXPLORER

OPEN EDITORS
- Welcome
- AI_COURSE_PROJECT

AI COURSE PROJECT
- AI_COURSE_PROJECT

OUTLINE

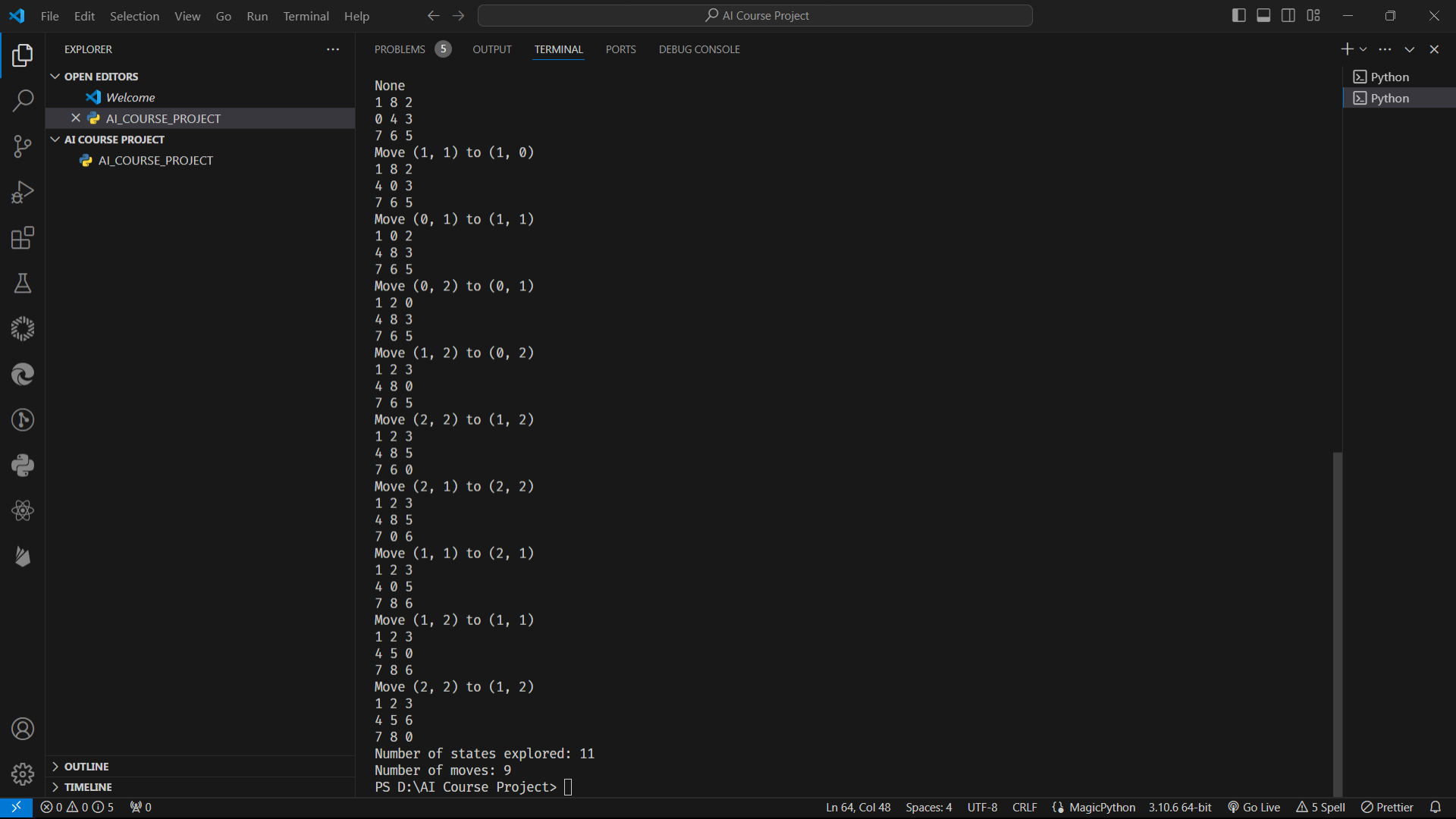
TIMELINE

PROBLEMS 5 OUTPUT TERMINAL PORTS DEBUG CONSOLE

```
PS D:\AI Course Project> & C:/Users/Abdelaleem/AppData/Local/Programs/Python/Python310/python.exe "d:/AI Course Project/AI_COURSE_PROJECT"
BFS Solution:
None
1 8 2
0 4 3
7 6 5
Move (1, 1) to (1, 0)
1 8 2
4 0 3
7 6 5
Move (0, 1) to (1, 1)
1 0 2
4 8 3
7 6 5
Move (0, 2) to (0, 1)
1 2 0
4 8 3
7 6 5
Move (1, 2) to (0, 2)
1 2 3
4 8 0
7 6 5
Move (2, 2) to (1, 2)
1 2 3
4 8 5
7 6 0
Move (2, 1) to (2, 2)
1 2 3
4 8 5
7 0 6
Move (1, 1) to (2, 1)
1 2 3
4 0 5
7 8 6
Move (1, 2) to (1, 1)
1 2 3
4 5 0
7 8 6
Move (2, 2) to (1, 2)
1 2 3
4 5 6
7 8 0
```

Python
Python

Ln 64, Col 48 Spaces: 4 UTF-8 CRLF MagicPython 3.10.6 64-bit Go Live 5 Spell Prettier


File Edit Selection View Go Run Terminal Help
AI Course Project
EXPLORER
OPEN EDITORS
Welcome
AI_COURSE_PROJECT
AI COURSE PROJECT
AI_COURSE_PROJECT
PROBLEMS 5
OUTPUT
TERMINAL
PORTS
DEBUG CONSOLE
None
1 8 2
0 4 3
7 6 5
Move (1, 1) to (1, 0)
1 8 2
4 0 3
7 6 5
Move (0, 1) to (1, 1)
1 0 2
4 8 3
7 6 5
Move (0, 2) to (0, 1)
1 2 0
4 8 3
7 6 5
Move (1, 2) to (0, 2)
1 2 3
4 8 0
7 6 5
Move (2, 2) to (1, 2)
1 2 3
4 8 5
7 6 0
Move (2, 1) to (2, 2)
1 2 3
4 8 5
7 0 6
Move (1, 1) to (2, 1)
1 2 3
4 0 5
7 8 6
Move (1, 2) to (1, 1)
1 2 3
4 5 0
7 8 6
Move (2, 2) to (1, 2)
1 2 3
4 5 6
7 8 0
Number of states explored: 11
Number of moves: 9
PS D:\AI Course Project>
Python
Python
OUTLINE
TIMELINE
Ln 64, Col 48
Spaces: 4
UTF-8
CRLF
MagicPython
3.10.6 64-bit
Go Live
5 Spell
Prettier