```
board = [ ]
for i in range (9):
    board.append (' ')

def printXO (letter, pos):
    board [pos] = letter

def empty (pos):
    if (board [pos] == ' '):
        return True
    else
        return False

def printBoard (board):
    for i in range (3):
        print ( board [3 * i] + '|' + board [3*i+1] + '|' +
                board [3*i+2])

def winning (board, l)
    return ( board [0] == l and board [1] == l and board [2] == l)
    or if (board [3] == l and board [4] == l and board [5] == l)
    or if (board [6] == l and board [7] == l and board [8] == l)
    or if (board [0] == l and board [3] == l and board [6] == l)
    or if (board [1] == l and board [4] == l and board [7] == l)
    or if (board [2] == l and board [5] == l and board [8] == l)
    or (board [0] == l and board [4] == l and board [8] == l)
    or (board [2] == l and board [4] == l and board [6] == l)

def Moves ():
    run = True
    while (run):
        move = int (input ("Select position for X :"))
        try:
            if (move >= 0 and move < 9):
```

```
            if empty(move):
                run = False.
                printXO('x', move)

            else
                print ("Invalid position")

        except:
            print(" please type number again:")

    printBoard(board)
    if(winning(board, 'x') == True)
            return "winner is x"
    if ( BoardFull(board)):
            return 'No winner"

    run = True

    While (run):
        move = int (input ("Select position for O :")
        - —
        - —
        - - -

    same for user 'O'

def BoardFull(board):

    if " in board:
        return false
    else:
        return True.
```

No algorithm

Algo:

(1) Take a board list created with a elements of ' '

(2) printXO() to add letter in pos.

(3) empty() to check if position is empty

(4) printBoard() to print the board.

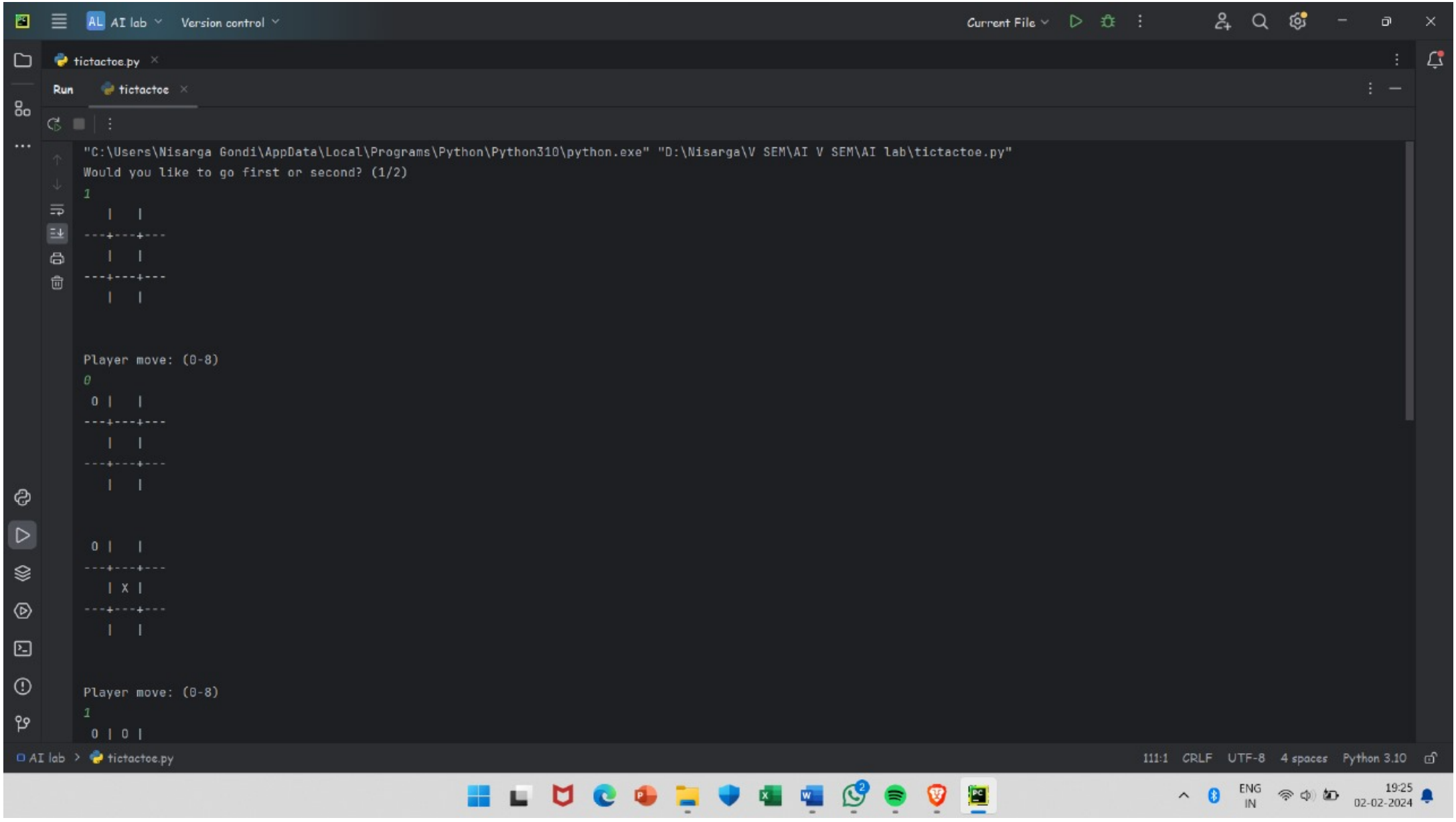
(5) winning() checking 8 conditions to win

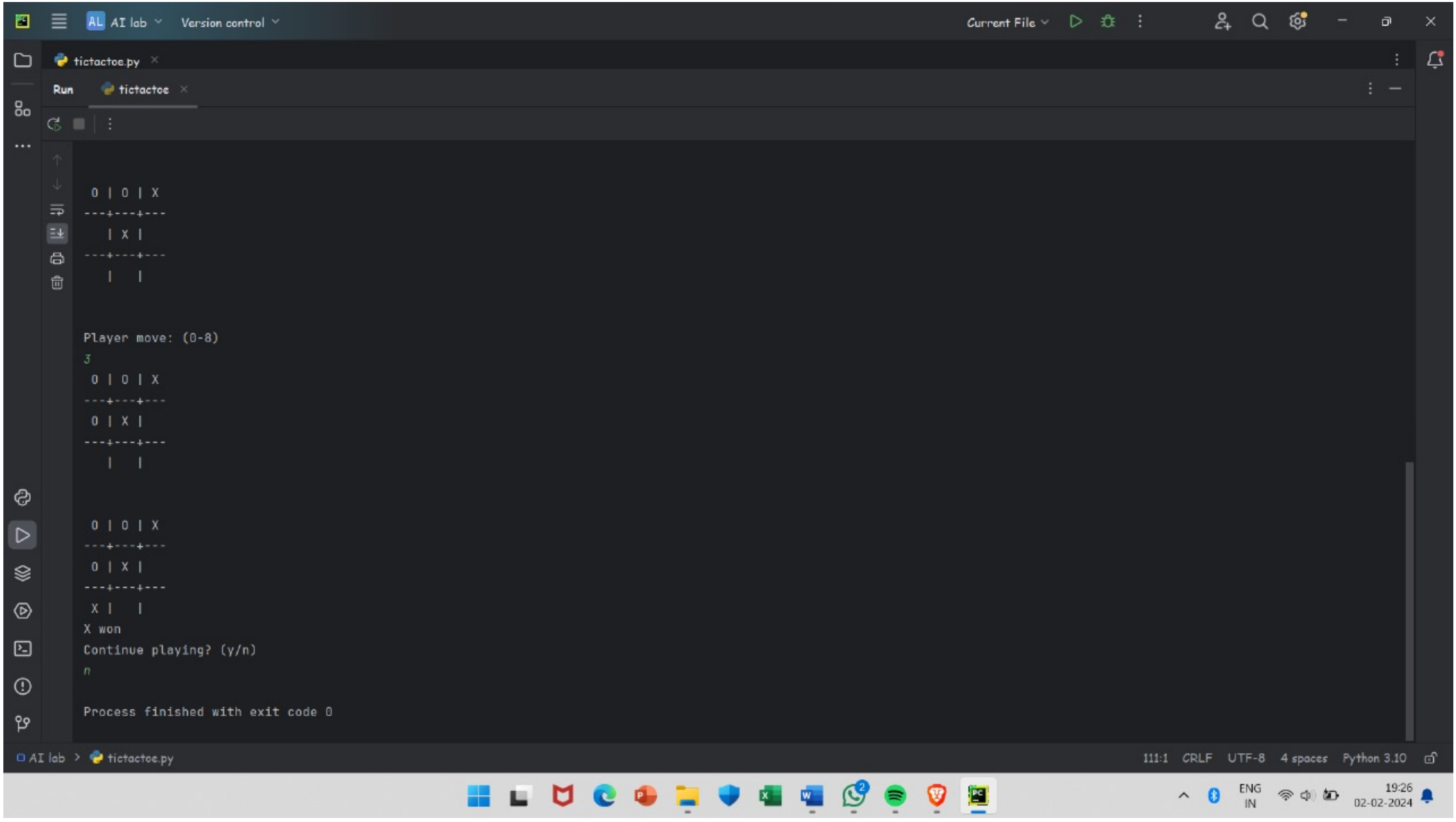
(6) Moves() function to input

- (i) X → check empty().
  - → print XO()
  - Check winning()
  - Check boardfull()
- (ii) O → check empty()
  - → print XO()

(7) BoardFull() to check if ' ' is more in board.

(8) main() to all Moves() till BoardFull()


AI lab
Version control
Current File
tictactoe.py
Run
tictactoe
"C:\Users\Nisarga Gondi\AppData\Local\Programs\Python\Python310\python.exe" "D:\Nisarga\V SEM\AI V SEM\AI lab\tictactoe.py"
Would you like to go first or second? (1/2)
1
Player move: (0-8)
0
Player move: (0-8)
1
AI lab > tictactoe.py
111:1 CRLF UTF-8 4 spaces Python 3.10
ENG IN
19:25
02-02-2024


AI lab
Version control
Current File
tictactoe.py
Run
tictactoe
O | O | X
---+---+---
| X |
---+---+---
| |
Player move: (0-8)
3
O | O | X
---+---+---
O | X |
---+---+---
| |
O | O | X
---+---+---
O | X |
---+---+---
X | |
X won
Continue playing? (y/n)
n
Process finished with exit code 0
AI lab > tictactoe.py
111:1 CRLF UTF-8 4 spaces Python 3.10
ENG
IN
19:26
02-02-2024

Lab - 3

8 puzzle problem using BFS :-

Algorithm :-

(i) first we need a function to print board.

i.e
```
def print_board(board):
    for row in board:
        print(row)
print()
```

(ii) next we need to give initial & goal states to determine least number of moves

(iii) call a function to solve the puzzle.

(iv) we convert the initial state into tuples (immutable) and add it to a variable let's say visited, and use a queue for BFS, queue will have initial state & initial path

```
while not queue.empty():
    current_state, path = queue.get()
    if current state == goal_state:
        return path
```

(v) Check for position of blank space using a get_blank_pos() function.

```
def get_blank_pos(board):
    for i in range(3):
        for j in range(3):
            if board[i][j] == 0:
                return i, j
```

(vi) now we have four possible moves for it to move i.e left(0,-1) right(0,1) up(1,0) down(-1,0)
∴ we go through the moves and check if move is valid or not using a fn say if_valid(new_x, new_y) function

```
def if_valid(x,y):
    return 0 <= x < 3 and 0 <= y < 3
```

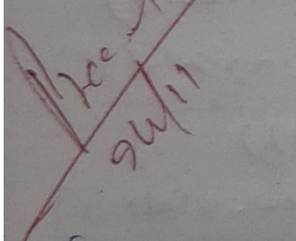
(vii) if move is valid we swap blank with the current pos.

ie if if_valid(newx, new_y)
new_state = copy the current board,
swap blank & curr pos and get new board

(viii) check if new_state not in visited.
add it to the visited.
update the queue with. new state.
path incremented.

(ix) after the for loop ends, if no solution is found return none.

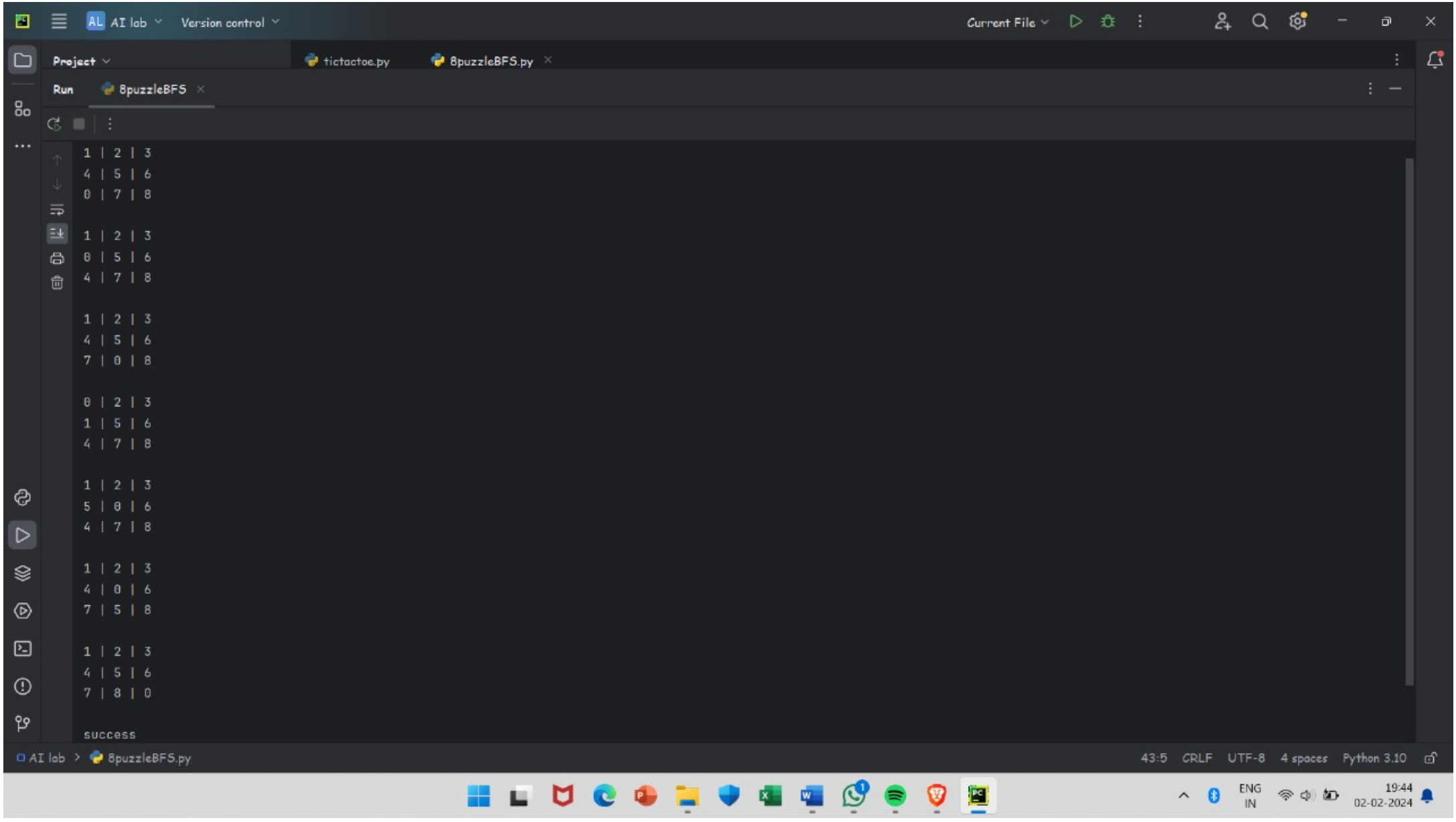
(x) print solution if moves ≥ 0.
else print no solution.



from queue import Queue

```
def get_blank_position(board):
    for i in range(3):
        for j in range(3):
            if board[i][j] == 0:
                return i, j

def is_valid_move(x, y):
    return 0 <= x < 3 and 0 <= y < 3

def swap(board, x1, y1, x2, y2):
    board[x1][y1], board[x2][y2] = board[x2][y2],
                                   board[x1][y1]

def solve_puzzle(initial_state, goal_state):
    visited = set()
    queue = Queue()
    queue.put((initial_state, 0))
```

```
    while not queue.empty():
        current_state, move_count = queue.get()
        visited.add(tuple(map(tuple, current_state)))

        if current_state == goal_state:
            return move_count

        blank_x, blank_y = get_blank_position(current_state)

        moves = [(0,1), (1,0), (0,-1), (-1,0)]

        for move in moves:
            new_x, new_y = blank_x + move[0], blank_y + move[

            if is_valid_move(new_x, new_y):
                new_state = [row.copy() for row in current_state]
                swap(new_state, blank_x, blank_y, new_x, new_y)

                if tuple(map(tuple, new_state)) not in visited:
                    queue.put((new_state, move_count + 1))
                    visited.add(tuple(map(tuple, new_state)))

    return None

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

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

move_count = solve_puzzle(initial_state, goal_state)
```


AI lab
Version control
Current File
Project
tictactoe.py
8puzzleBFS.py
Run
8puzzleBFS
1 | 2 | 3
4 | 5 | 6
0 | 7 | 8
1 | 2 | 3
0 | 5 | 6
4 | 7 | 8
1 | 2 | 3
4 | 5 | 6
7 | 0 | 8
0 | 2 | 3
1 | 5 | 6
4 | 7 | 8
1 | 2 | 3
5 | 0 | 6
4 | 7 | 8
1 | 2 | 3
4 | 0 | 6
7 | 5 | 8
1 | 2 | 3
4 | 5 | 6
7 | 8 | 0
success
AI lab > 8puzzleBFS.py
43:5 CRLF UTF-8 4 spaces Python 3.10
ENG
IN
19:44
02-02-2024

8 - Puzzle Iterative Deepening Search:

① define the goal state using a function.

```
goalstate = [ [ ],
              [ ],
              [ ]
            ]
```

② to use Iterative Deepening search we will use the algorithm to recursively call a function until we get the solution & return none if answer not there.

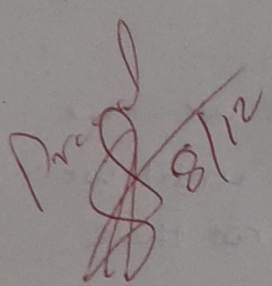
```
def depth_limited_search(state, parent=None, action=None, depth=0,
                         depth_limit=0):
    if is_goal(state):
        return parent, action
    elif depth == depth_limit
        return none
    else:
        for a in get_actions(state):
            new_state = apply_action(state, a)
            result = depth_limited_search(new_state, state, a, depth+1,
                                          depth_limit)
            if result is not None:
                return result
        return None
def iterative_deepy(initial_state):
    depth_limit = 0
    while True:
        result = depth_limited_search(initial state, depth_lim = depth_lim)
        if result is not None:
            return result
        depth_limit += 1
```

③ write a function to get all the possible actions

```
actions = []
blank_row, blank_col = find_blank(state)
if blank_row > 0
    actions.append("up")
if blank_row < 2
    actions.append("down")
if blank_col > 0
    actions.append("left")
if blank_col < 2
    action.append("right")
```

④ to find blank position we gonna use.

```
for i in range(3):
  for j in range(3):
    if state[i][j] == 0:
      return i, j
```



Code:

```
MAX_DEPTH = 2
def id_dfs(puzzle, goal, get_moves):
    import itertools
    def dfs(route, depth):
        if depth == 0:
            return
        if route[-1] == goal:
            return route
        for move in get_moves(route[-1]):
            if move not in route:
                next_route = dfs(route + [move], depth-1)
                if next_route:
                    return next_route
    for depth in itertools.count():
        route = dfs([puzzle], depth)
        if route:
            return route
def possible_moves(state):
    b = state.index(0)
    d = []
    if b not in [0,1,2]:
        d.append('u')
    if b not in [6,7,8]:
        d.append('d')
```

```
    if b not in [0, 3, 6]:
        d.append('l')
    if b not in [2, 5, 8]:
        d.append('r')

    pos_moves = []
    for i in d:
        pos_moves.append(generate(state, i, b))
    return pos_moves

def generate(state, m, b):
    temp = state.copy()
    if m == 'd':
        temp[b+3], temp[b] = temp[b], temp[b+3]
    if m == 'u':
        temp[b-3], temp[b] = temp[b], temp[b-3]
    if m == 'l':
        temp[b-1], temp[b] = temp[b], temp[b-1]
    if m == 'r':
        temp[b+1], temp[b] = temp[b], temp[b+1]
    return temp

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

route = id_dfs(initial, goal, possible_moves)
if route:
    print("It is possible to solve the puzzle")
    print("Path:", route)
else:
    print("Failed to find a solution")
```

output:

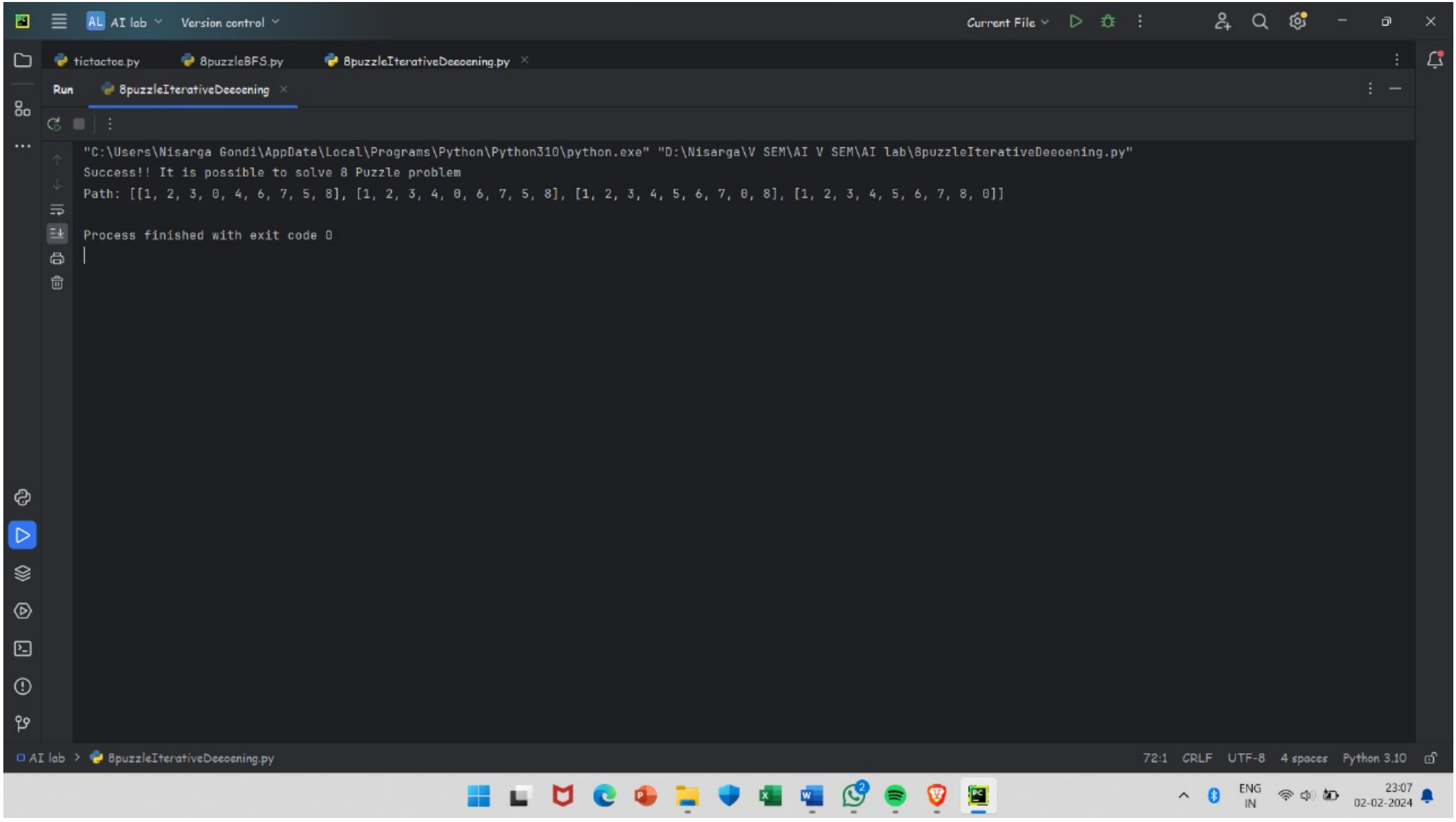
'8 puzzle quiz solveable'

Path : [1, 2, 3, 0, 4, 6, 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, 7, 8, 0]

Print depth

Print < fail
Pass

8/12


AI lab
Version control
Current File
tictactoe.py
8puzzleBFS.py
8puzzleIterativeDeepening.py
Run
8puzzleIterativeDeepening
"C:\Users\Nisarga Gondi\AppData\Local\Programs\Python\Python310\python.exe" "D:\Nisarga\V SEM\AI V SEM\AI lab\8puzzleIterativeDeepening.py"
Success!! It is possible to solve 8 Puzzle problem
Path: [[1, 2, 3, 0, 4, 6, 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]]
Process finished with exit code 0
AI lab > 8puzzleIterativeDeepening.py
72:1 CRLF UTF-8 4 spaces Python 3.10
ENG IN
23:07
02-02-2024

8 puzzle using A*.

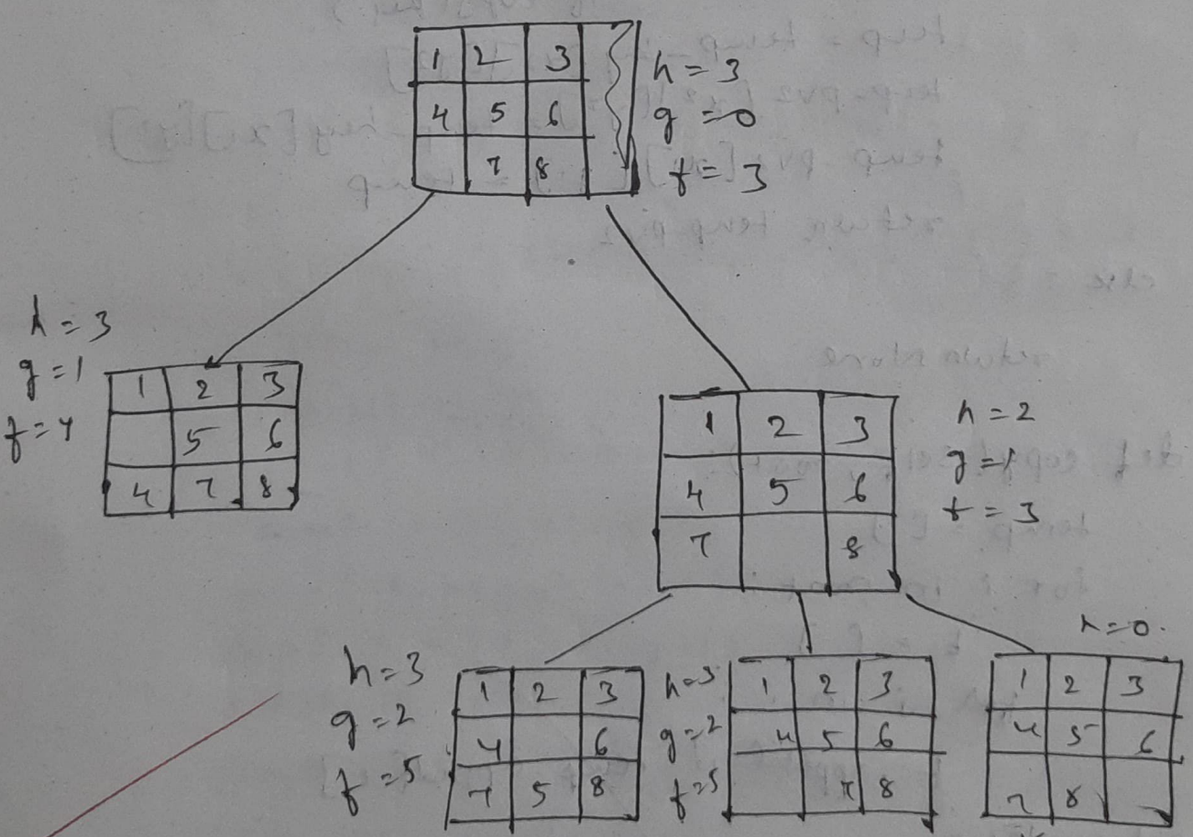
Algorithm

1. Create initial and goal state for problem.
2. We calculate value at each step

   fvalue = hvalue + pathcost

3. Go to the path having least fvalue.
4. Goal state is reached when fvalue = 0
5. $f(h) = h(x) + g(x)$.

   $f(h)$ → heuristic cost; $h(x)$ → goal cost; $g(x)$ → path cost

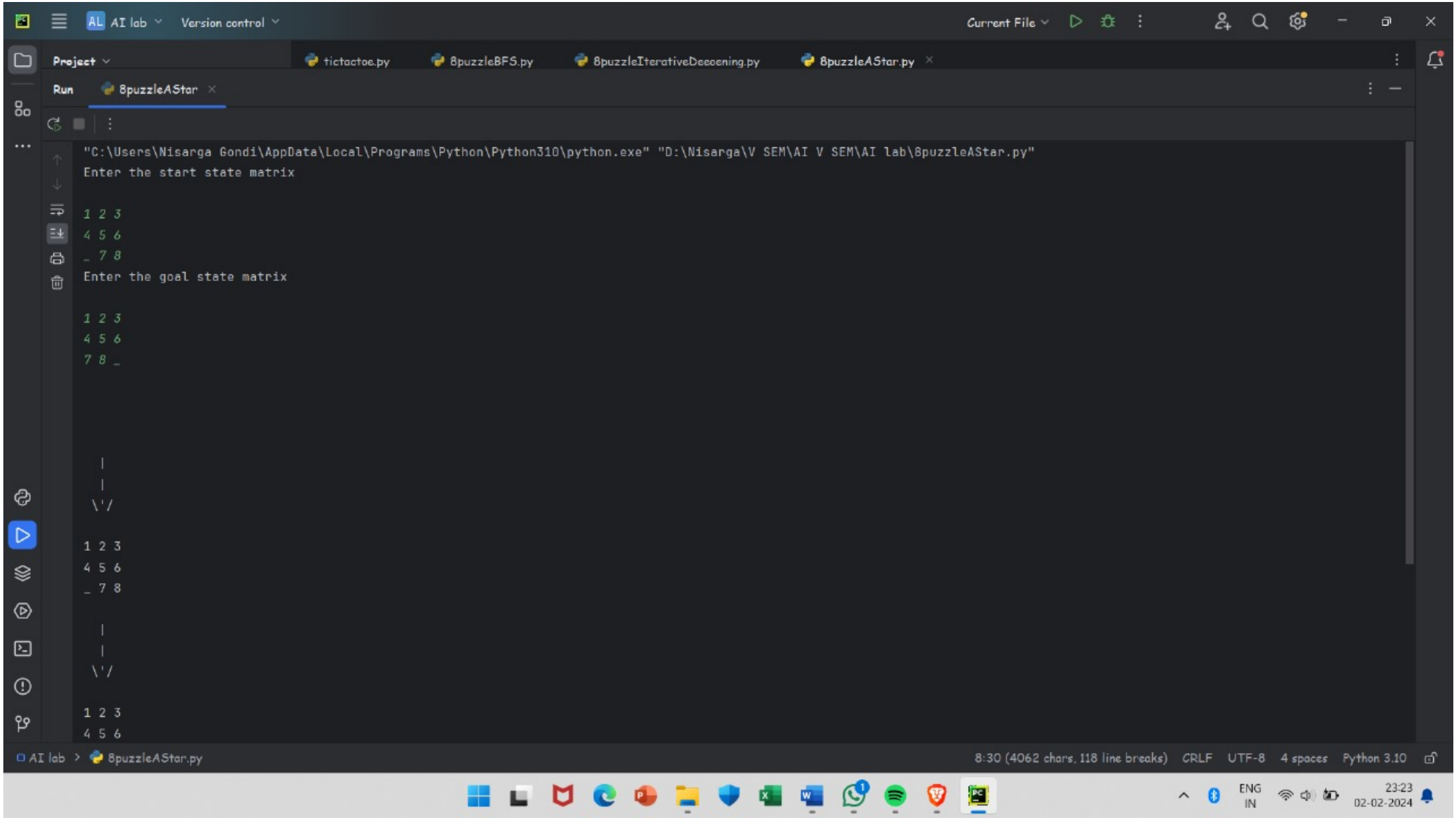


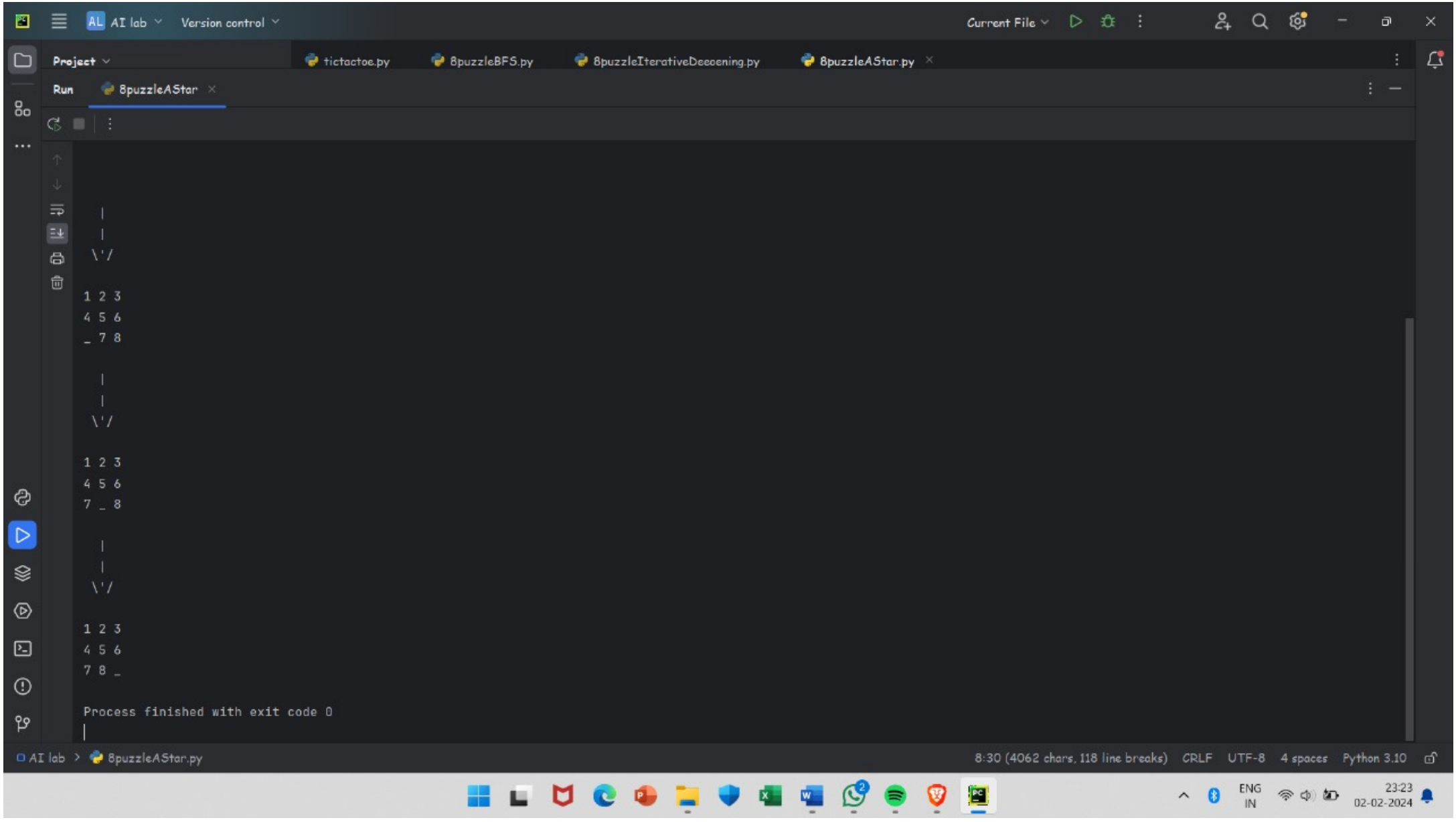
Code:

```
Class Node:
    def __init(self, data, level, fval)
        self.data = data.
        self.level = level
        self.fval = fval.
```

```
def generate_child(self):
    x, y = self.find(self.data, '-')
    val_list = [[x, y-1], [x, y+1], [x-1, y], [x+1, y]]
    children = []
    for i in val_list:
        child = self.shuffle(self.data, x, y, i[0], i[1])
        if child is not None:
            childNode = Node(child, self.level+1, 0)
            children.append(childNode)
    return children

def shuffle(self, key, x1, y1, x2, y2):
    if x2 >= 0 and x2 < len(self.data) and y2 >= 0 and
       y2 < len(self.data):
        temp_PV3 = []
        temp_PV3 = self.copy(key)
        temp = temp_PV3[x2][y2]
        temp_PV3[x2][y2] = temp_PV3[x1][y1]
        temp_PV3[x1][y1] = temp
        return temp_PV3
    else:
        return None

def copy(self, root):
    temp = []
    for i in root:
        t = []
        for j in i:
            t.append(j)
        temp.append(t)
    return temp

def find(self, PV3, x):
    for i in range(0, len(self.data)):
        for j in range(0, len(self.data)):
            if PV3[i][j] == x:
                return i, j
```

```
class puzzle:
    def __init__(self, size):
        self.n = size
        self.open = []
        self.closed = []
    def accept(self):
        puz = []
        for i in range(0, self.n):
            temp = input().split(" ")
            puz.append(temp)
        return puz
    def f(self, start, goal)
        return self.h(start.data, goal) + start
    def h(self, start, goal):
        temp = 0
        for i in range(0, self.n):
            for j in range(0, self.n):
                if start[i][j] != goal[i][j] and
                                    start[i][j] != 
        return temp
    def process(self):
        print("Enter the start state matrix \n");
        start = self.accept()
        print("Enter goal state matrix \n")
        goal = self.accept()
        start = Node
        start.goal = self.f(start, goal)
        self.open.append(start)
        print("\n\n")
        while(True):
            cur = self.open[0]
            print("\n")
            for i in data:
                for j in i:
                    print(j, end = " ")
```


AI lab
Version control
Current File
Project
tictactoe.py
8puzzleBFS.py
8puzzleIterativeDeepening.py
8puzzleAStar.py
Run
8puzzleAStar
"C:\Users\Nisarga Gondi\AppData\Local\Programs\Python\Python310\python.exe" "D:\Nisarga\V SEM\AI V SEM\AI lab\8puzzleAStar.py"
Enter the start state matrix
1 2 3
4 5 6
_ 7 8
Enter the goal state matrix
1 2 3
4 5 6
7 8 _
|
|
\'/
1 2 3
4 5 6
_ 7 8
|
|
\'/
1 2 3
4 5 6
AI lab > 8puzzleAStar.py
8:30 (4062 chars, 118 line breaks) CRLF UTF-8 4 spaces Python 3.10
ENG
IN
23:23
02-02-2024


AI lab
Version control
Current File
Project
tictactoe.py
8puzzleBFS.py
8puzzleIterativeDeepening.py
8puzzleAStar.py
Run
8puzzleAStar
|
|
\'/
1 2 3
4 5 6
_ 7 8
|
|
\'/
1 2 3
4 5 6
7 _ 8
|
|
\'/
1 2 3
4 5 6
7 8 _
Process finished with exit code 0
AI lab > 8puzzleAStar.py
8:30 (4062 chars, 118 line breaks) CRLF UTF-8 4 spaces Python 3.10
ENG
IN
23:23
02-02-2024

Vacuum cleaner problem.

Step1: Initialize Rooms:

Create room with names, size and grid representing dirty(1) and clean(0) positions.

room_a = { 'name' : 'A', 'Right, clean or dirty }

Step2: If Status = Dirty then clean.

else if location = A and status = clean then return right

else if location = B and status = clean then return left

else exit

Step3: If both the locations are clear the vacuum cleaner is done with its task.

(OR)

Step1: Initialize rooms:

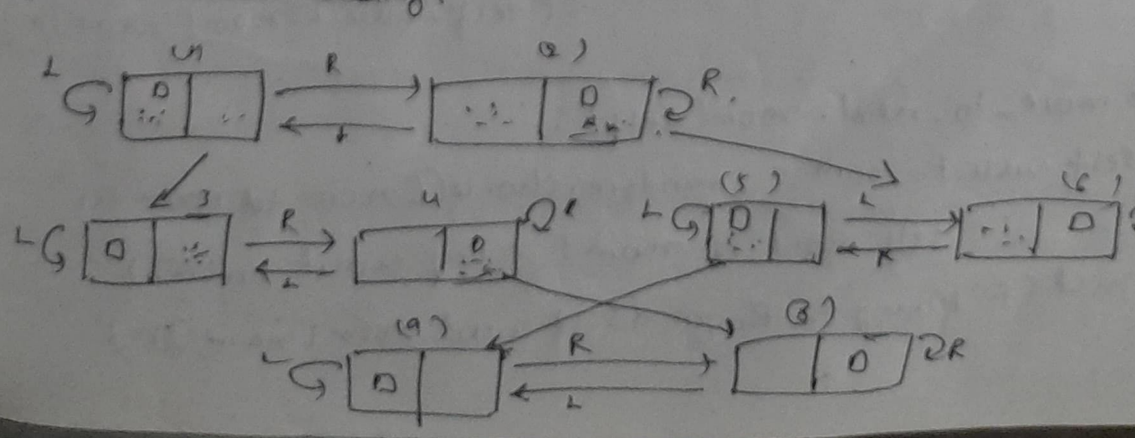
Create rooms with name, size and grid representing dirty & clean.

room_a = { 'name' : 'A', 'size' : (5,5), 'grid' : [[random.choice([0,1]) for _ in range(5)] for _ in range(5)]}

Step2: Initialize vacuum cleaner.

vacuum = VacuumCleaner(rooms)

Step3: Simulate cleaning.



Code :

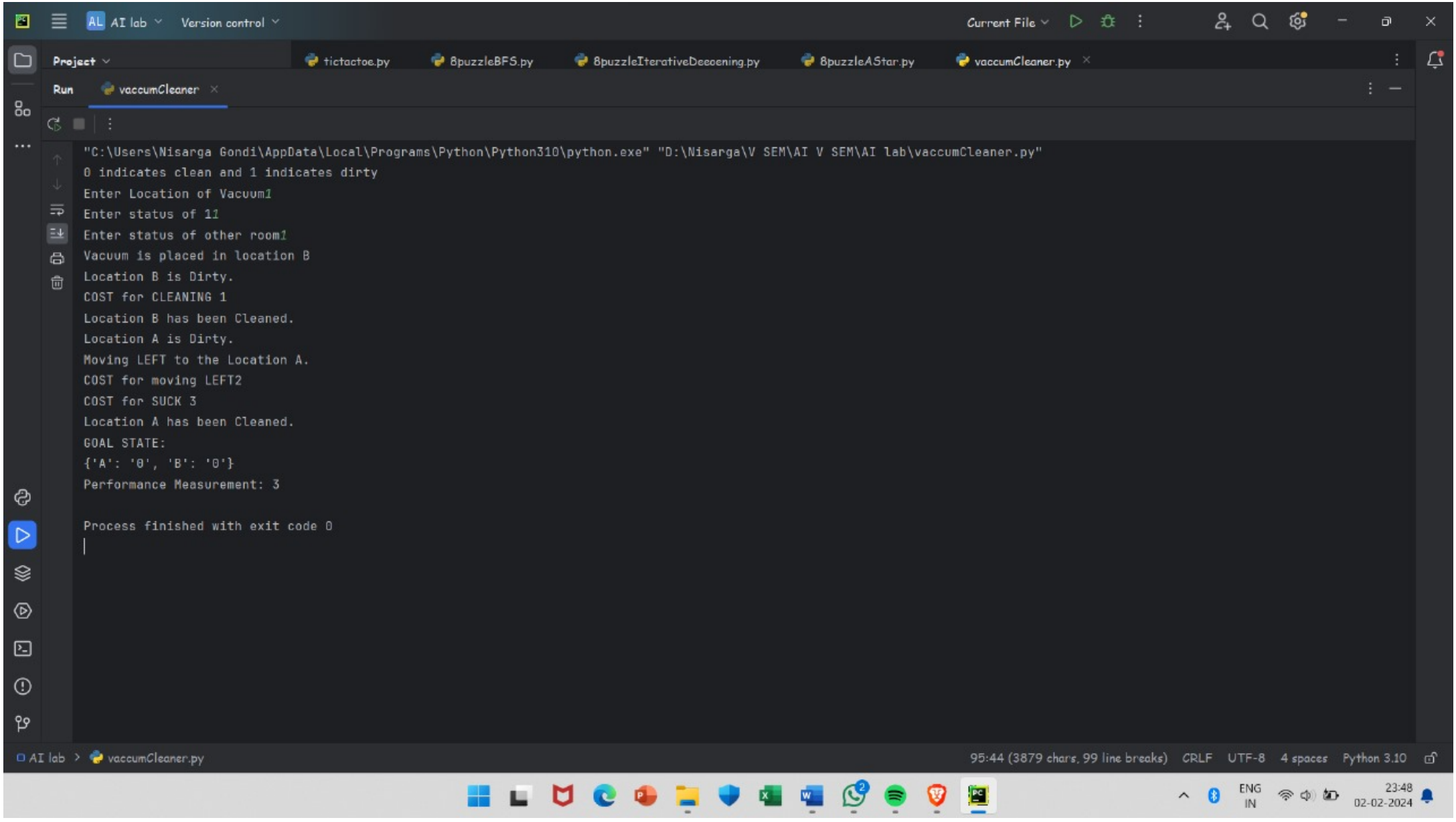
```
import random

class VacuumCleaner:
    def __init__(self, rooms):
        self.rooms = rooms
        self.current_room = random.choice(rooms)
        self.position = [0, 0]

    def move_random(self):
        direction = random.choice(['up', 'down', 'left', 'right'])
        if direction == 'up' and self.position[1] < self.current_room
                ['size'][1] - 1:
            self.position[1] += 1
        elif direction == 'down' and self.position[1] > 0:
            self.position[1] -= 1
        elif direction == 'left' and self.position[0] > 0:
            self.position[0] -= 1
        elif direction == 'right' and self.position[0] < self.current_room
                ['size'][0] - 1:
            self.position[1] += 1

    def clean(self):
        x, y = self.position
        if self.current_room['grid'][y][x] == 1:
            print(f"Cleaning dirty position {self.position} in Room
                {self.current_room['name']}.")
            self.current_room['grid'][y][x] = 0
        else:
            print(f"Already clean position {self.position} in Room
                {self.current_room['name']}")

    def move_to_next_room(self):
        self.current_room = random.choice([room for room in
            self.rooms if room != self.current_room])
        print(f"Moving to Room {self.current_room['name']}")
```

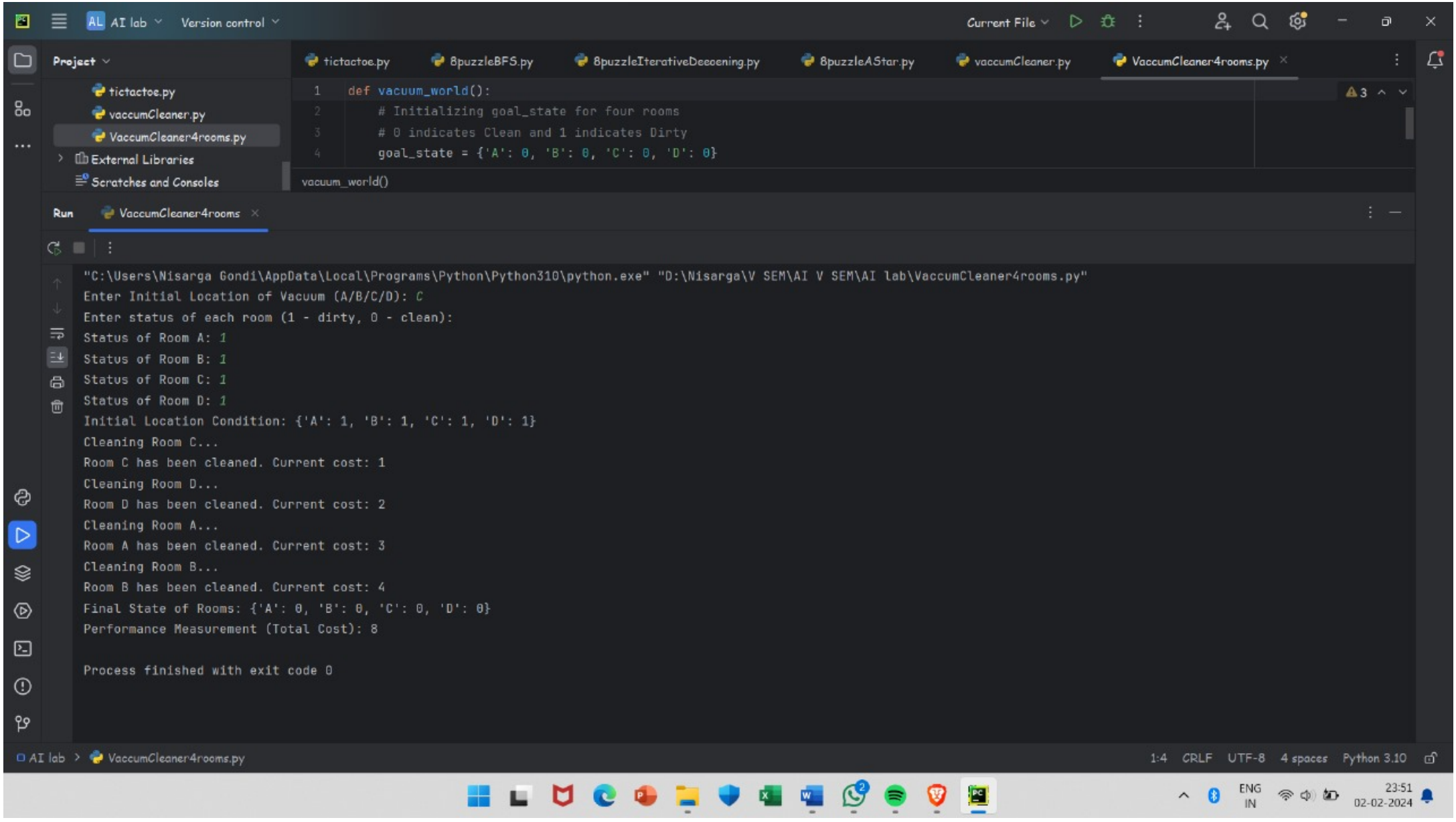
```
def simulate_vacuum_cleaner(rooms):
    vacuum = VacuumCleaner(rooms)
    for room in rooms:
        vacuum.current_room = room.
        for y in range(room['size'][1]):
            for x in range(room['size'][0]):
                vacuum.position = [x,y]
                vacuum.clean()
                vacuum.move_random()
        vacuum.move_to_next_room()

if __name__ == "__main__":
    room_a = {'name': 'A', 'size': (5,5), 'grid':
                [[random.choice([0,1]) for _ in range(
                    for _ in range(5)]}
    room_b = {"name": 'B', 'size': (4,4), 'grid':
                [[random.choice([0,1]) for _ in range(4)] for _ in
                    range(4)]}
    print(room_a['grid'])
    print(room_b['grid'])
    rooms = [room_a, room_b]
    simulate_vacuum_cleaner(rooms)
```

<u>output</u>.

```
room A: [[0,1],[1,0]]
room B: [[1,0]]
Already clean position [0,0] in Room A
Cleaning dirty position [1,0] in Room A
Cleaning dirty position [0,1] in Room A
Already clean position [1,1] in Room A
Moved to Room B
Cleaning dirty position [0,0] in Room B
Already clean position [1,0] in Room B
Moving to Room A
```

22/12


AI lab
Version control
Current File
Project
tictactoe.py
8puzzleBFS.py
8puzzleIterativeDeepening.py
8puzzleAStar.py
vaccumCleaner.py
Run
vaccumCleaner
"C:\Users\Nisarga Gondi\AppData\Local\Programs\Python\Python310\python.exe" "D:\Nisarga\V SEM\AI V SEM\AI lab\vaccumCleaner.py"
0 indicates clean and 1 indicates dirty
Enter Location of Vacuum1
Enter status of 11
Enter status of other room1
Vacuum is placed in location B
Location B is Dirty.
COST for CLEANING 1
Location B has been Cleaned.
Location A is Dirty.
Moving LEFT to the Location A.
COST for moving LEFT2
COST for SUCK 3
Location A has been Cleaned.
GOAL STATE:
{'A': '0', 'B': '0'}
Performance Measurement: 3
Process finished with exit code 0
AI lab > vaccumCleaner.py
95:44 (3879 chars, 99 line breaks)
CRLF
UTF-8
4 spaces
Python 3.10
ENG
IN
23:48
02-02-2024


AI lab
Version control
Current File
Project
tictactoe.py
vaccumCleaner.py
VaccumCleaner4rooms.py
External Libraries
Scratches and Consoles
tictactoe.py
8puzzleBFS.py
8puzzleIterativeDeepening.py
8puzzleAStar.py
vaccumCleaner.py
VaccumCleaner4rooms.py
def vacuum_world():
# Initializing goal_state for four rooms
# 0 indicates Clean and 1 indicates Dirty
goal_state = {'A': 0, 'B': 0, 'C': 0, 'D': 0}
vacuum_world()
Run
VaccumCleaner4rooms
"C:\Users\Nisarga Gondi\AppData\Local\Programs\Python\Python310\python.exe" "D:\Nisarga\V SEM\AI V SEM\AI lab\VaccumCleaner4rooms.py"
Enter Initial Location of Vacuum (A/B/C/D): C
Enter status of each room (1 - dirty, 0 - clean):
Status of Room A: 1
Status of Room B: 1
Status of Room C: 1
Status of Room D: 1
Initial Location Condition: {'A': 1, 'B': 1, 'C': 1, 'D': 1}
Cleaning Room C...
Room C has been cleaned. Current cost: 1
Cleaning Room D...
Room D has been cleaned. Current cost: 2
Cleaning Room A...
Room A has been cleaned. Current cost: 3
Cleaning Room B...
Room B has been cleaned. Current cost: 4
Final State of Rooms: {'A': 0, 'B': 0, 'C': 0, 'D': 0}
Performance Measurement (Total Cost): 8
Process finished with exit code 0
AI lab > VaccumCleaner4rooms.py
1:4 CRLF UTF-8 4 spaces Python 3.10
ENG IN
23:51 02-02-2024

Knowledge Based Entailment

Algorithm:

1) Create knowledge Base:
   Define propositional symbols P, Q and R.
   Use logical statements to define knowledge base.

```
knowledge_base = And ( Implies(P,Q),
                       Implies(Q,R),
                       Not(R)
                     )
```

2) Define Query.
   Define the query using a propositional symbol.

```
query = Symbols('P')
```

3) Check entailment.
   Use the 'satisfiable' function to check if the knowledge base entails the negation of the query.

```
entailment = satisfiable( And (knowledge_base, Not(query)))
```

4) If there is no satisfying assignment (i.e the knowledge base entails the query)
   print "The query is logically entailed by the knowledge base"
   If there is a satisfying assignment, print "The query is not logically entailed by the knowledge base".

Practical

Example:

P: It is raining.
Q: I will take an umbrella
R: I will stay dry.

Knowledge base
1) implies(P, Q)
2) implies(Q, R)
3) not(R)

Query
(P) It is raining True

rule
Will take umbrella, hence will stay dry.

conclusion entails.

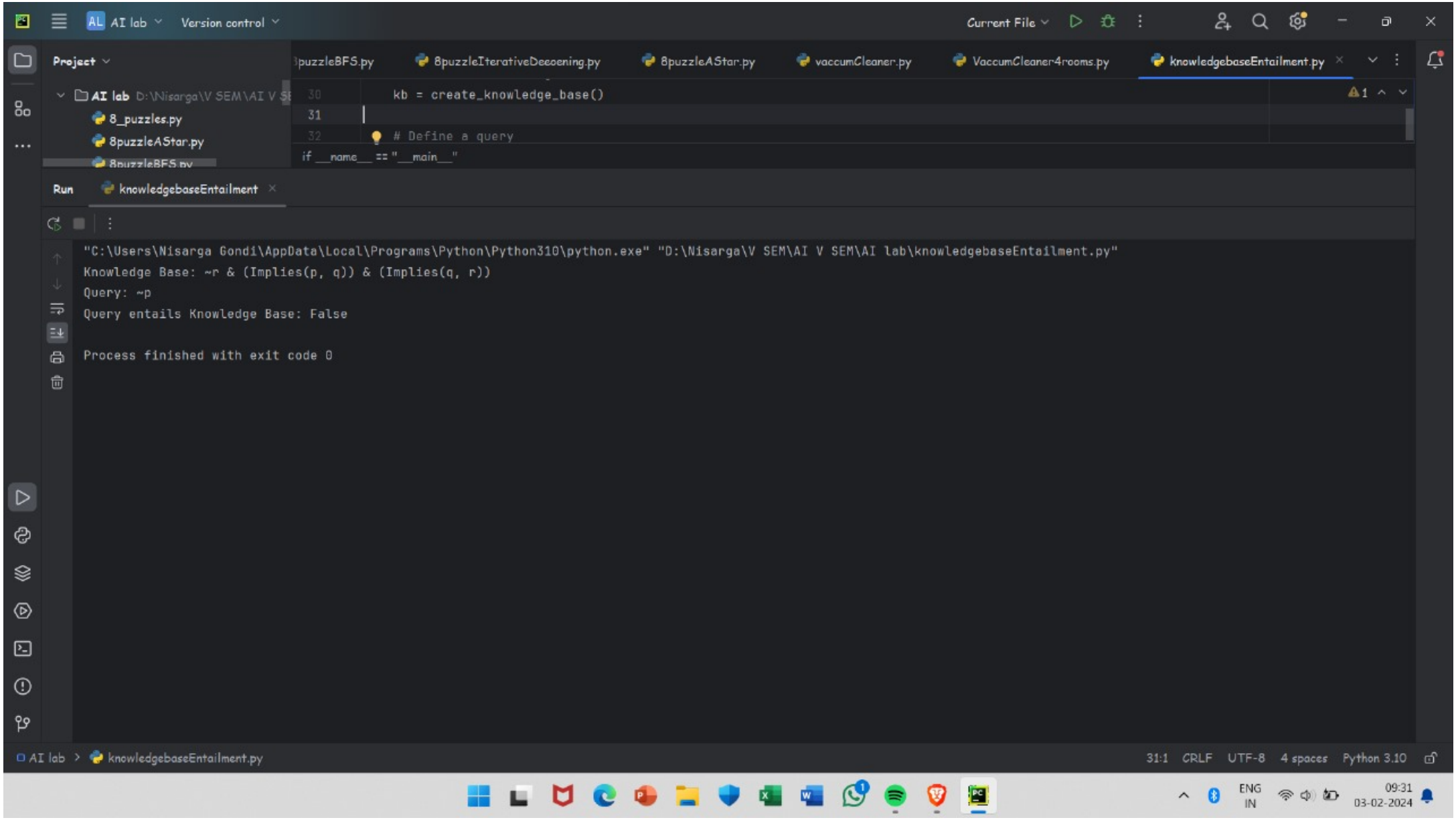
Code:

```
from sympy import symbols, And, Not, Implies, satisfiable

def create_knowledge_base():
    P = symbols('p')
    q = symbols('q')
    r = symbols('r')

    knowledge_base = And( Implies( p, q),
                          Implies( q, r),
                          Not(r)
                        )

    return knowledge_base

def query_entails ( knowledge_base, query):
    entailment = satisfiable( And ( knowledge_base , Not(query)))
    return not entailment

if __name__ == "__main__":
    kb = create_knowledge_base()
    query = symbols('p')
    result = query_entails (kb, query)
    print(" Knowledge Base :", kb)
    print ("Query:", query)
    Print ("Query entails Knowledge Base :", result)

def tell(kb, rule):
    kb.append(rule)

combinators =
```

Output:

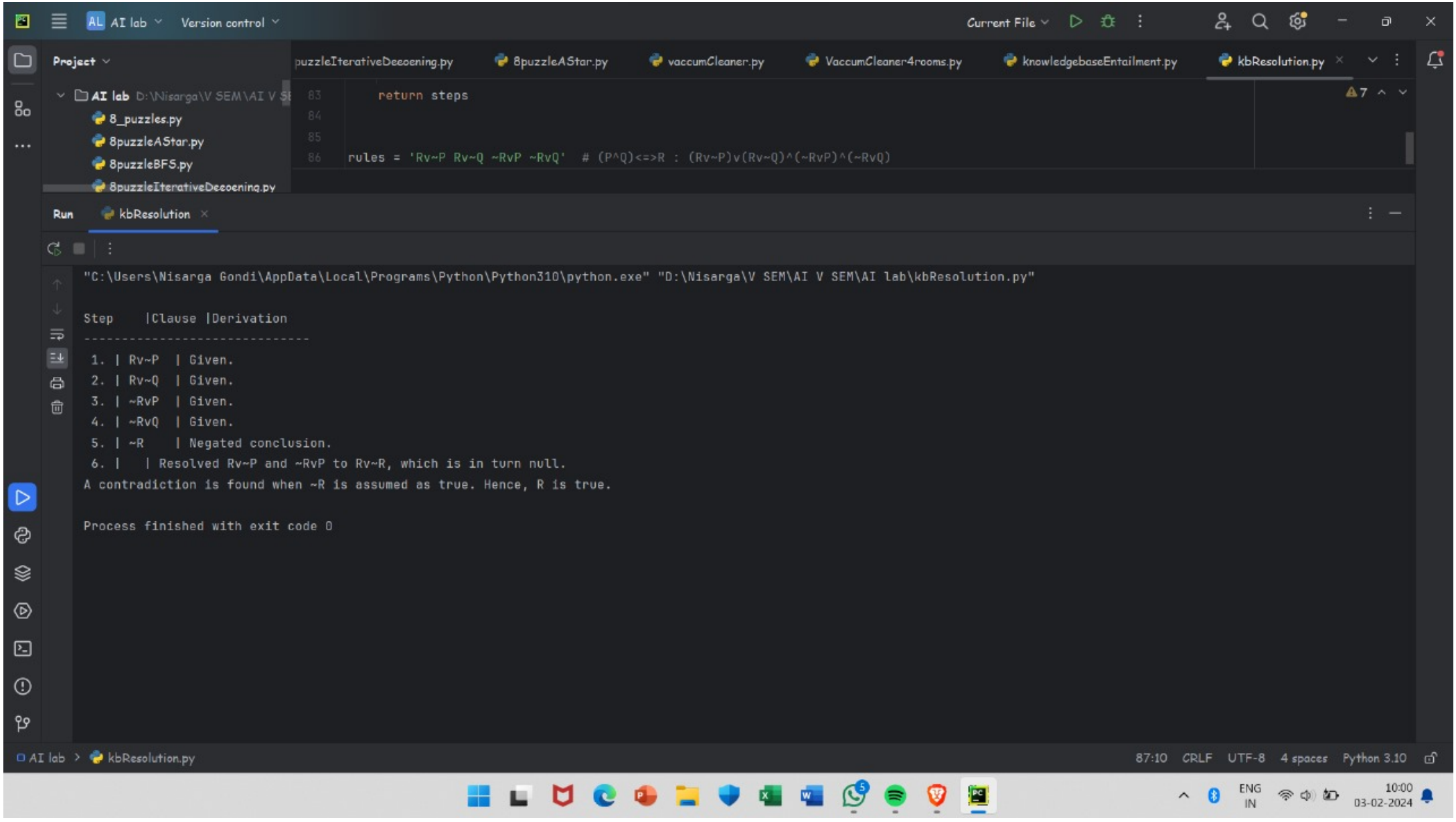
```
Knowledge Base : ~r & (Implies(p,q)) & (Implies(q,r))
Query : p
Query entails knowledge Base : False

Query : ~q
Query entails knowledge Base : False.
```

27/12/23


AI lab
Version control
Current File
Project
AI lab D:\Nisarga\V SEM\AI V SE
8_puzzles.py
8puzzleAStar.py
8puzzleBFS.py
puzzleBFS.py
8puzzleIterativeDeepening.py
8puzzleAStar.py
vaccumCleaner.py
VaccumCleaner4rooms.py
knowledgebaseEntailment.py
kb = create_knowledge_base()
# Define a query
if __name__ == "__main__"
Run
knowledgebaseEntailment
"C:\Users\Nisarga Gondi\AppData\Local\Programs\Python\Python310\python.exe" "D:\Nisarga\V SEM\AI V SEM\AI lab\knowledgebaseEntailment.py"
Knowledge Base: ~r & (Implies(p, q)) & (Implies(q, r))
Query: ~p
Query entails Knowledge Base: False
Process finished with exit code 0
AI lab > knowledgebaseEntailment.py
31:1 CRLF UTF-8 4 spaces Python 3.10
ENG
IN
09:31
03-02-2024

Knowledge Base Resolution:

```
def negate_literal(literal)
    if literal[0] == '~':
        return literal[1:]
    else:
        return '~' + literal

def resolve(c1, c2):
    resolved_clause = set(c1) | set(c2)

    for literal in c1:
        if negate_literal(literal) in c2:
            resolved_clause.remove(literal)
            resolved_clause.remove(negate_literal(literal))
    return tuple(resolved_clause)

def resolution(knowledge_base):
    while True:
        new_clause = set()
        for i, c1 in enumerate(knowledge_base):
            for j, c2 in enumerate(knowledge_base):
                if i != j:
                    new_clause = resolve(c1, c2)
                if len(new_clause) > 1 and new_clause not
                        in knowledge_base:
                    new_clause.add(new_clause)

        if not new_clause:
            break
        knowledge_base != new_clause
    return knowledge_base

if __name__ == "__main__":
    kb = {('p','q'), ('~p','r'), ('~q','~r')}
    result = resolution(kb)
    print("original kb", kb)
    print("resolved kb", result)
```


AI lab
Version control
Current File
Project
AI lab D:\Nisarga\V SEM\AI V SE
8_puzzles.py
8puzzleAStar.py
8puzzleBFS.py
8puzzleIterativeDeepening.py
puzzleIterativeDeepening.py
8puzzleAStar.py
vaccumCleaner.py
VaccumCleaner4rooms.py
knowledgebaseEntailment.py
kbResolution.py
return steps
rules = 'Rv~P Rv~Q ~RvP ~RvQ' # (P^Q)<=>R : (Rv~P)v(Rv~Q)^(~RvP)^(~RvQ)
Run
kbResolution
"C:\Users\Nisarga Gondi\AppData\Local\Programs\Python\Python310\python.exe" "D:\Nisarga\V SEM\AI V SEM\AI lab\kbResolution.py"
Step |Clause |Derivation
1. | Rv~P | Given.
2. | Rv~Q | Given.
3. | ~RvP | Given.
4. | ~RvQ | Given.
5. | ~R | Negated conclusion.
6. | | Resolved Rv~P and ~RvP to Rv~R, which is in turn null.
A contradiction is found when ~R is assumed as true. Hence, R is true.
Process finished with exit code 0
AI lab > kbResolution.py
87:10 CRLF UTF-8 4 spaces Python 3.10
ENG IN
10:00 03-02-2024

Unification

Eg: knows(John, x) knows(John, Jane)

{x/Jane}

Step 1: If term1 or term2 is a variable or constant then:

a) term1 or term2 are identical
return NIL

b) Else if term1 is available
if term1 occurs in term2 :
return FAIL
else
return {(term1/term2)}

c) else if term2 is a variable
if term2 occur in term1:
return FAIL
else
return {(term1/term2)}

d) else return FAIL

Step 2: If predicate(term1) ≠ predicate(term2)
return FAIL

Step 3: number of argument ≠
return FAIL

Step 4: set(SUBST) to NIL

Step 5: For i=1 to the number of elemet in term1

a) Call unify(ith term1, ith term2)
put result into S

b) S= FAIL
return FAIL

c) If S ≠ NIL
a. Apply S to the remainds of both L1 & L2
b. SUBST, APPEND(S, SUBST)

Step 6: Return SUBST

unification code :

```
import re

def getAttributes(expression):
    expression = expression.split('(')[1:]
    expression = '('.join(expression)
    expression = expression[:-1]
    expression = re.split('(?<!\(),(?!\))', expression)
    return expression

def getInitialPredicate(expression):
    return expression.split('(')[0]

def isConstant(char):
    return char.isupper() and len(char) == 1

def isVariable(char):
    return char.islower() and len(char) == 1

def replaceAttributes(exp, old, new):
    attributes = getAttributes(exp)
    for index, val in enumerate(attributes):
        if val == old:
            attributes[index] = new
    predicate = getInitialPredicate(exp)
    return predicate + "(" + ",".join(attributes) + ")"

def apply(exp, substitutions):
    for substitution in substitutions:
        new, old = substitution
        exp = replaceAttributes(exp, old, new)
    return exp

def checkOccurs(var, exp):
    if exp.find(var) == -1:
        return False
    return True

def getFirstRest(expression):
    attributes = getAttributes(expression)
    return attributes[0]
```

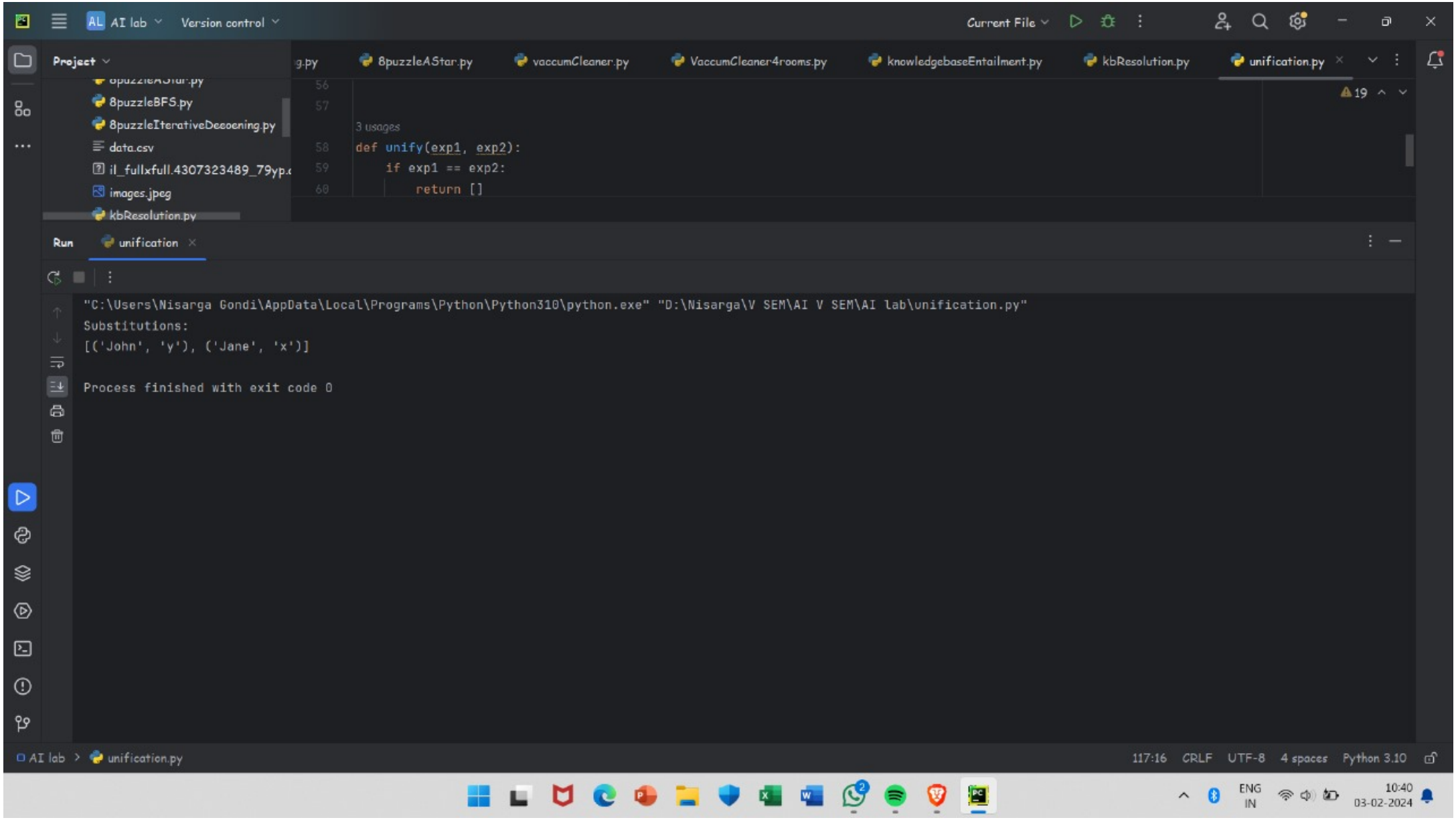
```
def getRemainingPart(expression):
    predicate = getInitialPredicate(expression)
    attributes = getAttributes(expression)
    newExpression = predicate + "(" + ",".join(Attributes[1:]) + ")"
    return newExpression

def unify(exp1, exp2):
    if exp1 == exp2:
        return []
    if isConstant(exp1) and isConstant(exp2):
        if exp1 != exp2:
            return False.
    if isConstant(exp1):
        return [(exp1, exp2)]
    if isConstant(exp2):
        return [(exp2, exp1)]
    if isVariable(exp1):
        if checkOccurs(exp1, exp2):
            return False.
        else:
            return [(exp2, exp1)]
    if isVariable(exp2):
        if checkOccurs(exp2, exp1):
            return False
        else:
            return [(exp1, exp2)]
    if getInitialPredicate(exp1) != getInitialPredicate(exp2):
        print("Predicates do not match. Cannot be unified")
        return False

attributeCount1 = len(getAttributes(exp1))
attributeCount2 = len(getAttributes(exp2))
if attributeCount1 != attributeCount2:
    return False.
head1 = getFirstPart(exp1)
head2 = getFirstPart(exp2)
initialSubstitution = unify(head1, head2)
if not initialSubstitution:
    return False
```

```
if attribute(exp1) == 1:
    return initialSubstitution.
tail1 = getRemainingPart(exp1)
tail2 = getRemainingPart(exp2)

if initialSubstitution != []:
    tail1 = apply(tail1, initialSubstitution)
    tail2 = apply(tail2, initialSubstitution)

remainingSubstitution = unify(tail1, tail2)
if not remainingSubstitution:
    return False.
initialSubstitution.extend(remainingSubst.)
return initialSubstitution.

exp1 = "knows(A, x)"
exp2 = "knows(y, mother(y))"
Substitutions = unify(exp1, exp2)
print("Substitutions:")
print(Substitutions)
```

output

Substitutions:

[('A', 'y'), ('x', 'x')]


AI lab
Version control
Current File
Project
8puzzleBFS.py
8puzzleIterativeDeepening.py
data.csv
images.jpeg
kbResolution.py
8puzzleAStar.py
vaccumCleaner.py
VaccumCleaner4rooms.py
knowledgebaseEntailment.py
kbResolution.py
unification.py
19
3 usages
def unify(exp1, exp2):
if exp1 == exp2:
return []
Run
unification
"C:\Users\Nisarga Gondi\AppData\Local\Programs\Python\Python310\python.exe" "D:\Nisarga\V SEM\AI V SEM\AI lab\unification.py"
Substitutions:
[('John', 'y'), ('Jane', 'x')]
Process finished with exit code 0
AI lab > unification.py
117:16 CRLF UTF-8 4 spaces Python 3.10
ENG
IN
10:40
03-02-2024

FOL to CNF conversion:

Step1: remove_implications

return " or(not({part[0]}), {part[1]}) "

Step2: Apply_demorgans law

formula.replace ('NOT(AND : 'or(NOT)').replace(ingabon)
'And(not'))

Step3: distribute_quantifiers

formula.replace('Forall(; 'And(').replace
('Exits(; 'or(')

Step1: Create a list of SKOLEM_CONSTANTS

Step2: Find ∀, ∃

If the attributes are lower case, replace them with a skolem constant

remove used skolem constant or function from the list

If the attribute are both lowercase and uppercase replace the uppercase attribute with a skolem function

Step3: Replace ⇔ with '-'

transform - as Q ≡ (P ⇒ Q) ∧ (Q ⇒ P)

Step4: replace ⇒ with '-'

Step5: Apply demorgans law

replace ~[

as ~P & ~Q if(| was present)

replace ~[

as ~P | ~Q if(& was present)

replace ~ with ''

FOL to CNF code:

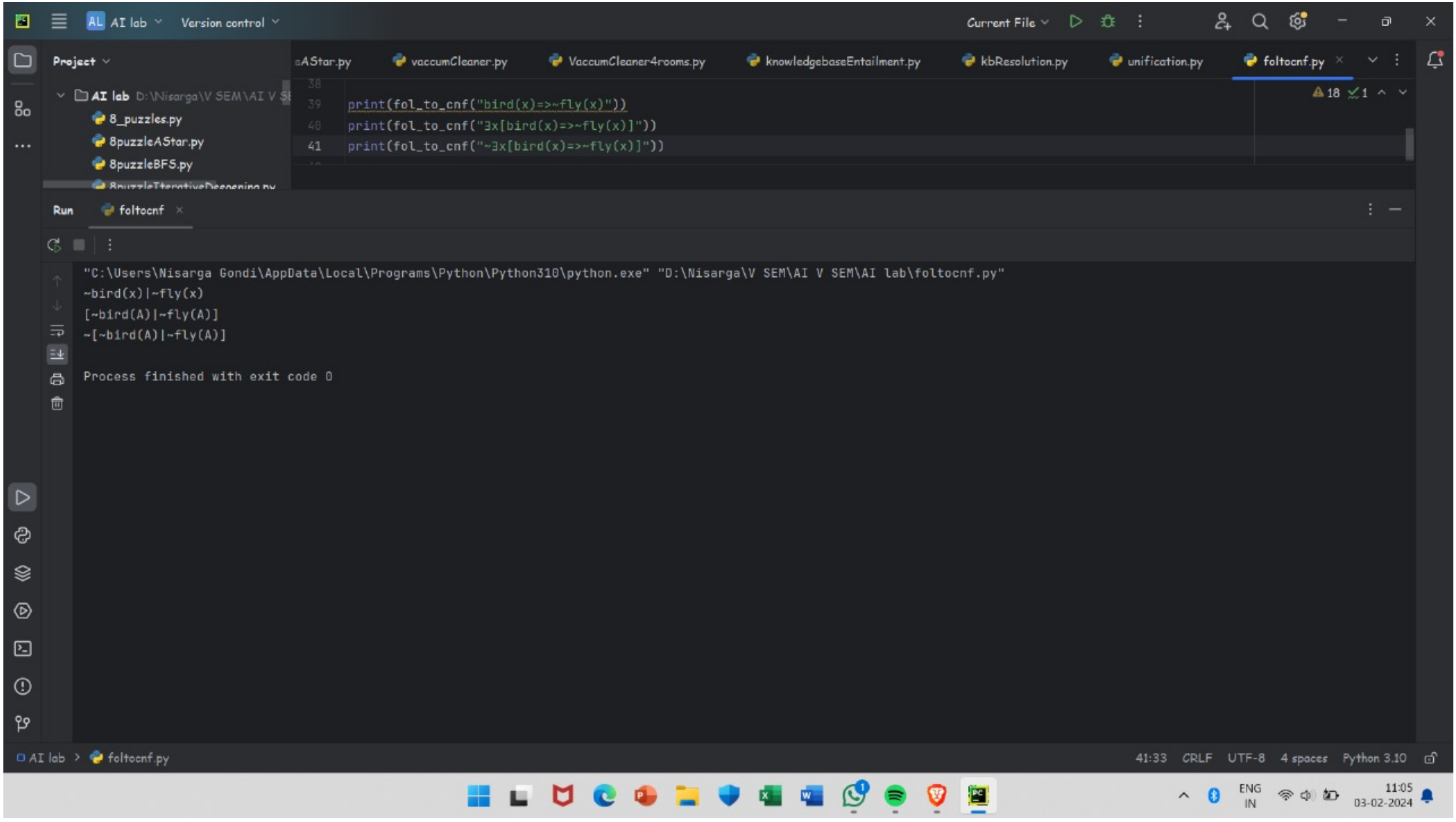
```
def getAttributes(string):
    expr = '\([^)]+\)'
    matches = re.findall(expr, string)
    return [m for m in str(matches) if m.isalpha()]
def getPredicates(string):
    expr = '[a-zA-Z~]+\([A-Za-z,]+\)'
    return re.findall(expr, string)
def DeMorgan(sentence):
    string = ''.join(list(sentence).copy())
    string = string.replace('~~', '')
    flag = '[' in string
    string = string.replace('~[', '')
    string = string.strip(']')
    for predicate in getPredicates(string):
        string = string.replace(predicate, f'~{predicate}')
    s = list(string)
    for i, c in enumerate(string):
        if c == '|':
            s[i] = '&'
        elif c == '&':
            s[i] = '|'
    string = ''.join(s)
    string = string.replace('~~', '')
    return f'[{string}]' if flag else string
def Skolemization(sentence):
    SKOLEM_CONSTANTS = [f'{chr(c)}' for c in range(ord('A'),
    statement = ''.join(list(sentence).copy())[ord('Z')+1)]
    matches = re.findall('[∀∃].', statement)
    for match in matches[::-1]:
        statement = statement.replace(match, '')
        statements = re.findall('\[[^\]]+\]', statement)
        for s in statements:
            statement = statement.replace(s, s[1:-1])
    elif match[1]=='∃':
    return statement
```

```
import re
def fol_to_cnf(fol):
    statement = fol.replace('<=>', '_')
    while '_' in statement:
        i = statement.index('_')
        new_statement = '[' + statement[:i] + '=>' + statement[i+1:] + ']&[' + statement[i+1:] + '=>' + statement[:i] + ']'
        statement = new_statement
    statement = statement.replace('=>', '-')
    expr = '\[([^\]]+)\]'
    statements = re.findall(expr, statement)
    for i, s in enumerate(statements):
        if '[' in s and ']' not in s:
            statements[i] += ']'
    for s in statements:
        statement = statement.replace(s, fol_to_cnf(s))
    while '~' in statement:
        i = statement.index('~')
        br = statement.index('[') if '[' in statement else 0
        new_statement = '~[' + statement[br:i] + '|' + statement[i+1:]
        statement = statement[:br] + new_statement if br > 0 else new_statement
    while '~∀' in statement:
        i = statement.index('~∀')
        statement = list(statement)
        statement[i], statement[i+1], statement[i+2] = '∃', statement[i+2], '~'
        statement = ''.join(statement)
    while '~∃' in statement:
        i = statement.index('~∃')
        s = list(statement)
        s[i], s[i+1], s[i+2] = '∀', s[i+2], '~'
        statement = ''.join(s)
        statement = statement.replace('~[∀', '[~∀')
        statement = statement.replace('~[∃', '[~∃')
    expr = '~\[([^]]+)\]'
    statements = re.findall(expr, statement)
    for s in statements:
        statement = statement.replace('~['+s+']', fol_to_cnf('~('+s+')'))
    return statement
```

```
for s in statements:
    statement = statement.replace(s, DeMorgan(s))
return statement.
print(skolemization(fol_to_cnf("animal(y)<=> loves(x,y)")))
print(skolemization(fol_to_cnf("∀x [∀y [animal(y) => loves
    (x,y)].] => [∃z [loves(z,x)]]")))
print(fol_to_cnf("[american(x) ∧ weapon(y) ∧ sells(x,y,z)
    ∧ hostile(z)] => criminal(x)"))
```

OUTPUT

```
[~animal(y) | loves(x,y)] & [~loves(x,y) | animal(y)].
[animal(G(x)) & ~loves(x, G(x))] | [loves(F(x),x)]
[~american(x) | ~weapon(y) | ~sells(x,y,z) | ~hostile(z) | criminal(x)]
```


AI lab
Version control
Current File
Project
AI lab D:\Nisarga\V SEM\AI V SE
8_puzzles.py
8puzzleAStar.py
8puzzleBFS.py
AStar.py
vaccumCleaner.py
VaccumCleaner4rooms.py
knowledgebaseEntailment.py
kbResolution.py
unification.py
foltocnf.py
print(fol_to_cnf("bird(x)=>~fly(x)"))
print(fol_to_cnf("∃x[bird(x)=>~fly(x)]"))
print(fol_to_cnf("~∃x[bird(x)=>~fly(x)]"))
Run
foltocnf
"C:\Users\Nisarga Gondi\AppData\Local\Programs\Python\Python310\python.exe" "D:\Nisarga\V SEM\AI V SEM\AI lab\foltocnf.py"
~bird(x)|~fly(x)
[~bird(A)|~fly(A)]
~[~bird(A)|~fly(A)]
Process finished with exit code 0
AI lab > foltocnf.py
41:33 CRLF UTF-8 4 spaces Python 3.10
ENG
IN
11:05
03-02-2024

forward Chaining:

```
import re
def isVariable(x):
    return len(x) ==1 and x.islower() and x.isalpha()

def getAttributes(string):
    expr = '\([^)]+\)'
    matches = re.findall(expr,string)
    return matches

def getPredicates(string):
    expr = '([a-zA-Z~]+)\([^&|]+\)'
    return re.findall(expr,string)

class Fact:
    def __init__(self,expression):
        self.expression = expression
        predicate, params = self.splitExpression(expression)
        self.predicate = predicate
        self.params = params
        self.result = any(self.getConstants())

    def splitExpression(self,expression):
        predicate = getPredicates(expression)[0]
        params = getAttributes(expression)[0].strip('()').split(',')

    def getResult(self):
        return self.result

    def getConstants(self):
        return [None if isVariable(c) else c for c in self.params]

    def getVariables(self):
        return [v if isVariable(v) else None for v in self.params]

    def substitute(self,constants):
        c = constants.copy()
        f = f"{self.predicate}({','.join([constants.pop(0) if
                                isVariable(p) else p for p in self.params])})"
        return Fact(f)
```

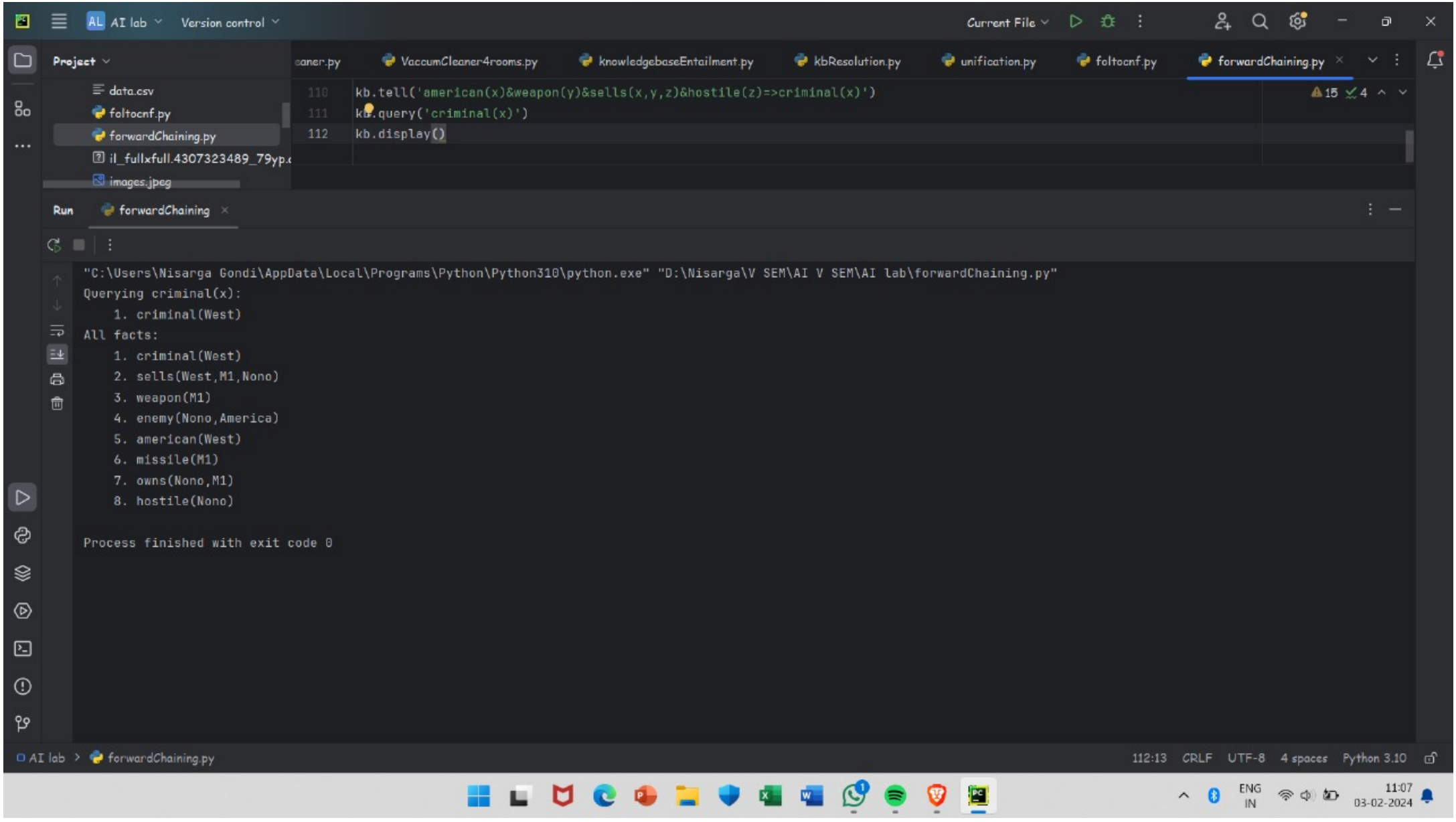
```
class Implication:
    def __init__(self, expression):
        self.expression = expression
        l = expression.split('=>')
        self.lhs = [Fact(f) for f in l[0].split('&')]

    def evaluate(self, facts):
        constants = {}
        new_lhs = []
        for fact in facts:
            for val in self.lhs:
                if val.predicate == fact.predicate:
                    for i, v in enumerate(val.getVariables()):
                        if v:
                            constants[v] = fact.getConstants()[i]
                    new_lhs.append(fact)
        predicate, attributes = getPredicates(self.rhs.expression)[0],
            str(getAttributes(self.rhs.expression)[0])
        for key in constants:
            if constants[key]:
                attributes = attributes.replace(key, constants[key])
                expr = f'{predicate}{attributes}'
                return Fact(expr) if len(new_lhs)

class KB:
    def __init__(self):
        self.facts = set()
        self.implications = set()

    def tell(self, e):
        if '=>' in e:
            self.implications.add(Implication(e))
        else:
            self.facts.add(Fact(e))
        for i in self.implications:
            res = i.evaluate(self.facts)
            if res:
                self.facts.add(res)
```

```
def query(self, e):
    facts = set([f.expression for f in self.facts])
    i = 1
    print(f'Query {e}:')
    for f in facts:
        if Fact(f).predicate == Fact(e).predicate:
            print(f'\t{i}. {f}')
            i += 1
def display(self):
    print("All facts:")
    for i, f in enumerate(set([f.expression for f in self.fa
        print(f'\t{i+1}. {f}')

kb = KB()
kb.tell('missile(x) => weapon(x)')
kb.tell('missile(M1)')
kb.tell('enemy(x, America) => hostile(x)')
kb.tell('american(West)')
kb.tell('missile(x) & owns(Nono, x) => sells(West, x, Nono)')
kb.tell('american(x) & weapon(y) & sells(x, y, z) & hostile(z)
                                    => criminal(x)')
kb.query('criminal(x)')
kb.display()
```

```
Query: criminal(x):
    1. criminal(West)
All facts:
    1. american(West)
    2. sells(West, M1, Nono)
    3. missile(M1)
    4. enemy(Nono, America)
    5. criminal(West)
    6. weapon(M1)
    7. owns(Nono, M1)
    8. hostile(Nono)
    9. criminal(West)
```

27/2/24

late submissions


AI lab
Version control
Current File
Project
data.csv
foltocnf.py
forwardChaining.py
images.jpeg
VaccumCleaner4rooms.py
knowledgebaseEntailment.py
kbResolution.py
unification.py
foltocnf.py
forwardChaining.py
kb.tell('american(x)&weapon(y)&sells(x,y,z)&hostile(z)=>criminal(x)')
kb.query('criminal(x)')
kb.display()
Run
forwardChaining
"C:\Users\Nisarga Gondi\AppData\Local\Programs\Python\Python310\python.exe" "D:\Nisarga\V SEM\AI V SEM\AI lab\forwardChaining.py"
Querying criminal(x):
1. criminal(West)
All facts:
1. criminal(West)
2. sells(West,M1,Nono)
3. weapon(M1)
4. enemy(Nono,America)
5. american(West)
6. missile(M1)
7. owns(Nono,M1)
8. hostile(Nono)
Process finished with exit code 0
AI lab > forwardChaining.py
112:13 CRLF UTF-8 4 spaces Python 3.10
ENG IN
11:07 03-02-2024