AI Applications

Upload screenshots of code and execution of 8-puzzle problem, Wumpus World, Vacuum cleaner, Sudoku and Crossword puzzle.

HU22CSEN0100287

Sai Ganesh Eswaraprasad

8 PUZZLE PROBLEM

```python
import copy
from heapq import heappush, heappop

n = 3
row = [1, 0, -1, 0]
col = [0, -1, 0, 1]

class priorityQueue:
    def __init__(self):
        self.heap = []

    def push(self, k):
        heappush(self.heap, k)

    def pop(self):
        return heappop(self.heap)

    def empty(self):
        return not self.heap

class node:
    def __init__(self, parent, mat, empty_tile_pos, cost, level):
        self.parent = parent
        self.mat = mat
        self.empty_tile_pos = empty_tile_pos
        self.cost = cost
        self.level = level

    def __lt__(self, nxt):
```

```
        return self.cost < nxt.cost

def calculateCost(mat, final) -> int:
    count = 0
    for i in range(n):
        for j in range(n):
            if mat[i][j] and mat[i][j] != final[i][j]:
                count += 1
    return count

def newNode(mat, empty_tile_pos, new_empty_tile_pos, level, parent, final) ->
node:
    new_mat = copy.deepcopy(mat)
    x1, y1 = empty_tile_pos
    x2, y2 = new_empty_tile_pos
    new_mat[x1][y1], new_mat[x2][y2] = new_mat[x2][y2], new_mat[x1][y1]
    cost = calculateCost(new_mat, final)
    return node(parent, new_mat, new_empty_tile_pos, cost, level)

def printMatrix(mat):
    for i in range(n):
        for j in range(n):
            print("%d " % (mat[i][j]), end=" ")
        print()

def isSafe(x, y):
    return 0 <= x < n and 0 <= y < n

def printPath(root):
    if root is None:
        return
    printPath(root.parent)
    printMatrix(root.mat)
    print()

def solve(initial, empty_tile_pos, final):
    pq = priorityQueue()
    cost = calculateCost(initial, final)
    root = node(None, initial, empty_tile_pos, cost, 0)
    pq.push(root)

    while not pq.empty():
        minimum = pq.pop()
        if minimum.cost == 0:
            printPath(minimum)
            return
        for i in range(4):
```

```
            new_tile_pos = [minimum.empty_tile_pos[0] + row[i],
minimum.empty_tile_pos[1] + col[i]]
            if isSafe(new_tile_pos[0], new_tile_pos[1]):
                child = newNode(minimum.mat, minimum.empty_tile_pos,
new_tile_pos, minimum.level + 1, minimum, final)
                pq.push(child)

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

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

empty_tile_pos = [1, 2]
solve(initial, empty_tile_pos, final)
```

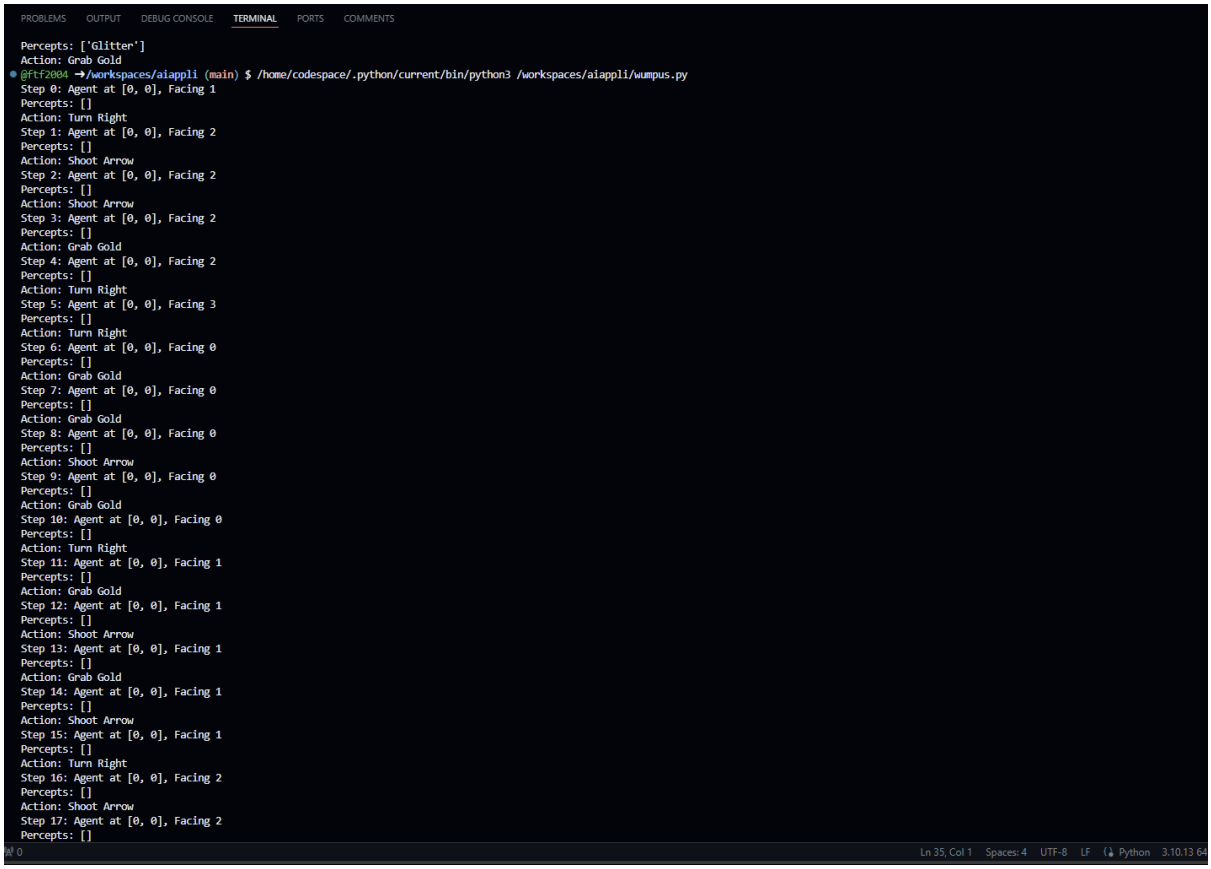
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

/home/codespace/.python/current/bin/python3 /workspaces/aiappli/8puzzle.py
@ftf2004 →/workspaces/aiappli (main) $ /home/codespace/.python/current/bin/python3 /workspaces/aiappli/8puzzle.py
1 2 3
5 6 0
7 8 4

1 2 3
5 0 6
7 8 4

1 2 3
5 8 6
7 0 4

1 2 3
5 8 6
0 7 4

@ftf2004 →/workspaces/aiappli (main) $
0
```

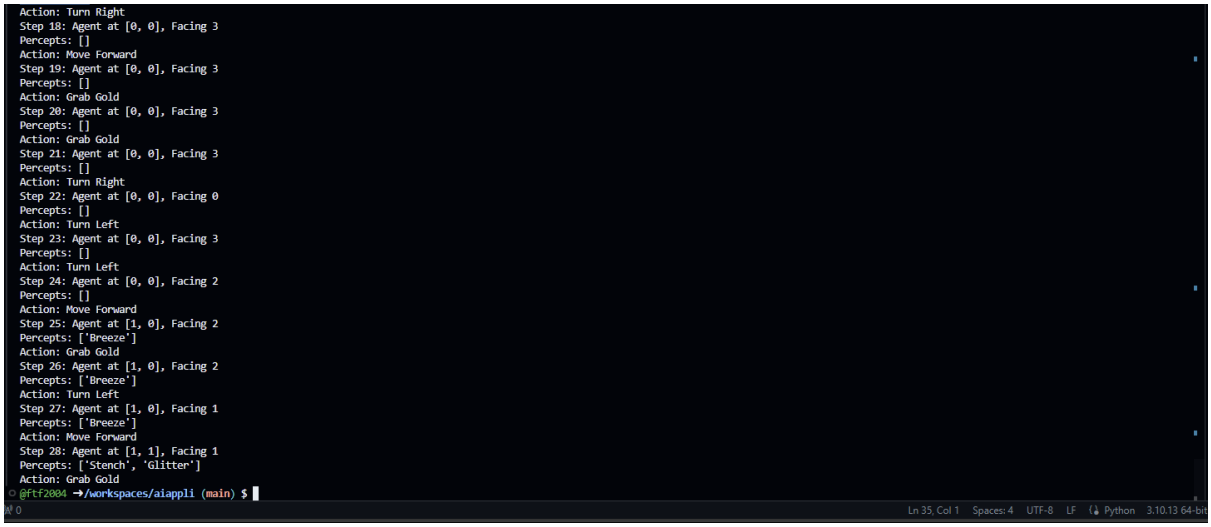
WUMPUS WORLD PROBLEM

```
                                import random

GRID_SIZE = 4

EMPTY, PIT, WUMPUS, GOLD, AGENT = 0, 1, 2, 3, 4

UP, RIGHT, DOWN, LEFT = 0, 1, 2, 3

class WumpusWorld:
    def __init__(self):
        self.grid = [[EMPTY for _ in range(GRID_SIZE)] for _ in
range(GRID_SIZE)]
        self.agent_position = [0, 0]
        self.agent_direction = RIGHT
        self.has_arrow = True
        self.has_gold = False

        for i in range(GRID_SIZE):
            for j in range(GRID_SIZE):
                if (i, j) != (0, 0) and random.random() < 0.2:
                    self.grid[i][j] = PIT

        self.grid[random.randint(1, GRID_SIZE - 1)][random.randint(1,
GRID_SIZE - 1)] = WUMPUS
        self.grid[random.randint(1, GRID_SIZE - 1)][random.randint(1,
GRID_SIZE - 1)] = GOLD

    def get_percepts(self):
        x, y = self.agent_position
        percepts = []
        if any(self.is_adjacent(x, y, WUMPUS)):
            percepts.append("Stench")
        if any(self.is_adjacent(x, y, PIT)):
            percepts.append("Breeze")
        if self.grid[x][y] == GOLD:
            percepts.append("Glitter")
        return percepts

    def is_adjacent(self, x, y, element):
        adjacent = []
        if x > 0:
            adjacent.append(self.grid[x - 1][y] == element)
        if x < GRID_SIZE - 1:
            adjacent.append(self.grid[x + 1][y] == element)
        if y > 0:
```

```
            adjacent.append(self.grid[x][y - 1] == element)
        if y < GRID_SIZE - 1:
            adjacent.append(self.grid[x][y + 1] == element)
        return adjacent

    def move_forward(self):
        x, y = self.agent_position
        if self.agent_direction == UP and x > 0:
            self.agent_position[0] -= 1
        elif self.agent_direction == DOWN and x < GRID_SIZE - 1:
            self.agent_position[0] += 1
        elif self.agent_direction == LEFT and y > 0:
            self.agent_position[1] -= 1
        elif self.agent_direction == RIGHT and y < GRID_SIZE - 1:
            self.agent_position[1] += 1

    def turn_left(self):
        self.agent_direction = (self.agent_direction - 1) % 4

    def turn_right(self):
        self.agent_direction = (self.agent_direction + 1) % 4

    def grab_gold(self):
        x, y = self.agent_position
        if self.grid[x][y] == GOLD:
            self.has_gold = True
            self.grid[x][y] = EMPTY

    def shoot_arrow(self):
        if self.has_arrow:
            self.has_arrow = False
            return "Scream"
        return None

def simulate():
    world = WumpusWorld()
    steps = 0

    actions = ["Move Forward", "Turn Left", "Turn Right", "Grab Gold", "Shoot
Arrow"]
    action_funcs = [world.move_forward, world.turn_left, world.turn_right,
world.grab_gold, world.shoot_arrow]

    while True:
        percepts = world.get_percepts()
        print(f"Step {steps}: Agent at {world.agent_position}, Facing
{world.agent_direction}")
        print("Percepts:", percepts)
```

```
        if "Glitter" in percepts:
            world.grab_gold()
            print("Action: Grab Gold")
            break

        if "Stench" in percepts and world.has_arrow:
            print("Action: Shoot Arrow")
            world.shoot_arrow()
        else:
            action = random.choice(action_funcs)
            action()
            print("Action:", actions[action_funcs.index(action)])

        steps += 1

        if steps > 100:
            print("Stopping simulation to prevent infinite loop.")
            break

simulate()
```




Action: Turn Right
Step 18: Agent at [0, 0], Facing 3
Percepts: []
Action: Move Forward
Step 19: Agent at [0, 0], Facing 3
Percepts: []
Action: Grab Gold
Step 20: Agent at [0, 0], Facing 3
Percepts: []
Action: Grab Gold
Step 21: Agent at [0, 0], Facing 3
Percepts: []
Action: Turn Right
Step 22: Agent at [0, 0], Facing 0
Percepts: []
Action: Turn Left
Step 23: Agent at [0, 0], Facing 3
Percepts: []
Action: Turn Left
Step 24: Agent at [0, 0], Facing 2
Percepts: []
Action: Move Forward
Step 25: Agent at [1, 0], Facing 2
Percepts: ['Breeze']
Action: Grab Gold
Step 26: Agent at [1, 0], Facing 2
Percepts: ['Breeze']
Action: Turn Left
Step 27: Agent at [1, 0], Facing 1
Percepts: ['Breeze']
Action: Move Forward
Step 28: Agent at [1, 1], Facing 1
Percepts: ['Stench', 'Glitter']
Action: Grab Gold
@ftf2004 →/workspaces/aiappli (main) $
Ln 35, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.13 64-bit

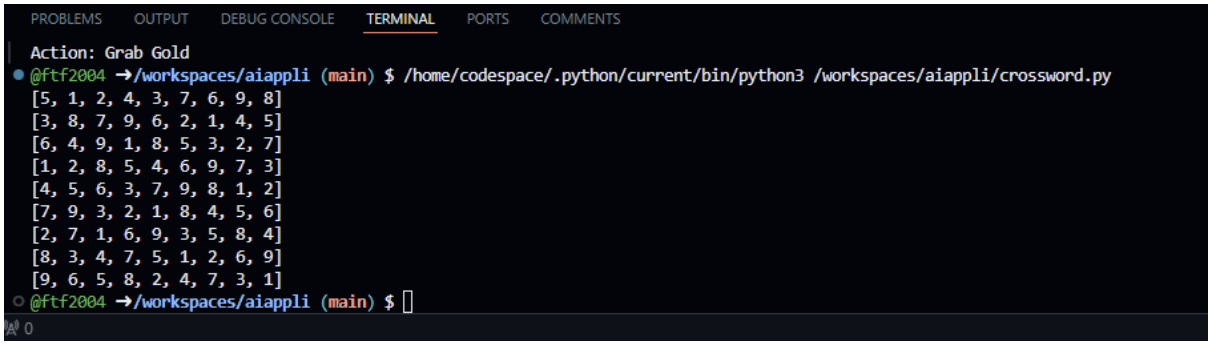
VACUUM CLEANER PROBLEM

```python
def vacuum_world():
    goal_state = {'A': '0', 'B': '0'}
    cost = 0

    loc = input("Enter Location of Vacuum: ")
    status = input("Enter status: ")
    otherstatus = input("Enter status of other room: ")

    if status == '1':
        print(f"Location {loc} is Dirty.")
        goal_state[loc] = '0'
        cost += 1
        print(f"Cost for CLEANING {loc} " + str(cost))

    if otherstatus == '1':
        otherloc = 'B' if loc == 'A' else 'A'
        print(f"Location {otherloc} is Dirty.")
        cost += 1
        print("Moving to other Location. Cost for moving " + str(cost))
        goal_state[otherloc] = '0'
        cost += 1
        print(f"Cost for CLEANING {otherloc}: " + str(cost))

    print("GOAL STATE: ")
    print(goal_state)
    print("Performance Measurement: " + str(cost))

vacuum_world()
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

/home/codespace/.python/current/bin/python3 /workspaces/ailab/optimised_vacuum23july.py
@ftf2004 ➜/workspaces/ailab (main) $ /home/codespace/.python/current/bin/python3 /workspaces/ailab/optimised_vacuum23july.py
Enter Location of Vacuum: A
Enter status: 0
Enter status of other room: 1
Location B is Dirty.
Moving to other Location. Cost for moving 1
Cost for CLEANING B: 2
GOAL STATE:
{'A': '0', 'B': '0'}
Performance Measurement: 2
@ftf2004 ➜/workspaces/ailab (main) $
```

SUDOKU

```python
def is_valid(board, row, col, num):
    if num in board[row]:
        return False

    if num in [board[i][col] for i in range(9)]:
        return False

    start_row, start_col = 3 * (row // 3), 3 * (col // 3)
    for i in range(start_row, start_row + 3):
        for j in range(start_col, start_col + 3):
            if board[i][j] == num:
                return False

    return True

def solve_sudoku(board):
    for row in range(9):
        for col in range(9):
            if board[row][col] == 0:
                for num in range(1, 10):
                    if is_valid(board, row, col, num):
                        board[row][col] = num
                        if solve_sudoku(board):
                            return True
                        board[row][col] = 0
                return False
    return True

sudoku_board = [
    [5, 0, 0, 0, 0, 0, 0, 0, 8],
    [0, 0, 0, 0, 6, 0, 0, 0, 0],
    [0, 4, 0, 0, 0, 0, 0, 2, 0],
    [0, 0, 8, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 3, 0, 9, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 4, 0, 0],
    [0, 7, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 5, 0, 0, 0, 0],
    [9, 0, 0, 0, 0, 0, 0, 0, 1]
]

if solve_sudoku(sudoku_board):
    for row in sudoku_board:
        print(row)
else:
    print("No solution exists")
```


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
COMMENTS
Action: Grab Gold
@ftf2004 →/workspaces/aiappli (main) $ /home/codespace/.python/current/bin/python3 /workspaces/aiappli/crossword.py
[5, 1, 2, 4, 3, 7, 6, 9, 8]
[3, 8, 7, 9, 6, 2, 1, 4, 5]
[6, 4, 9, 1, 8, 5, 3, 2, 7]
[1, 2, 8, 5, 4, 6, 9, 7, 3]
[4, 5, 6, 3, 7, 9, 8, 1, 2]
[7, 9, 3, 2, 1, 8, 4, 5, 6]
[2, 7, 1, 6, 9, 3, 5, 8, 4]
[8, 3, 4, 7, 5, 1, 2, 6, 9]
[9, 6, 5, 8, 2, 4, 7, 3, 1]
@ftf2004 →/workspaces/aiappli (main) $
0

CROSSWORD PUZZLE

```
def can_place_horizontally(grid, word, row, col):
    if col + len(word) > len(grid[0]):
        return False
    for i in range(len(word)):
        if grid[row][col + i] not in ('-', word[i]):
            return False
    return True

def can_place_vertically(grid, word, row, col):
    if row + len(word) > len(grid):
        return False
    for i in range(len(word)):
        if grid[row + i][col] not in ('-', word[i]):
            return False
    return True

def place_word(grid, word, row, col, direction):
    positions = []
    for i in range(len(word)):
        if direction == 'H':
            grid[row][col + i] = word[i]
            positions.append((row, col + i))
        else:  # direction == 'V'
            grid[row + i][col] = word[i]
            positions.append((row + i, col))
    return positions

def remove_word(grid, positions):
    for row, col in positions:
        grid[row][col] = '-'

def solve_crossword(grid, words, index):
    if index == len(words):
        return True
    word = words[index]
    for row in range(len(grid)):
        for col in range(len(grid[0])):
            if can_place_horizontally(grid, word, row, col):
                positions = place_word(grid, word, row, col, 'H')
                if solve_crossword(grid, words, index + 1):
                    return True
                remove_word(grid, positions)
            if can_place_vertically(grid, word, row, col):
                positions = place_word(grid, word, row, col, 'V')
                if solve_crossword(grid, words, index + 1):
```

```
                    return True
                remove_word(grid, positions)
    return False

def crossword_solver(grid, words):
    grid = [list(row) for row in grid]
    if solve_crossword(grid, words, 0):
        return [''.join(row) for row in grid]
    return None

# Example usage
grid = [
    "++++++++++-",
    "-++++++++-",
    "-------++-",
    "-++++++++-",
    "-++++++++-",
    "-++++-----",
    "------+++-",
    "-++++++++-",
    "+---------",
    "++++++++++"
]

words = ["CIVICS", "HISTORY", "MATH", "STAR", "PHYSICS", "CHEMISTRY"]

solved_grid = crossword_solver(grid, words)
if solved_grid:
    for row in solved_grid:
        print(row)
else:
    print("No solution exists")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS
@ftf2004 ➜/workspaces/aiappli (main) $ /home/codespace/.python/current/bin/python3 /workspaces/aiappli/crossword.py
++++++++C
P++++++++I
HISTORY++V
Y++++++++I
S++++++++C
I++++MATH-
CSTAR-+++-
S++++++++-
+CHEMISTRY
++++++++++
@ftf2004 ➜/workspaces/aiappli (main) $
```