**Artificial and Computational Intelligence Assignment 2**

Assignment members-

|  |  |  |  |
| --- | --- | --- | --- |
| Sl no | Name | BITS ID | Contribution Percentage |
| 1 | Chandupatla Anirudh Reddy | 2022da04387 | 100% |
| 2 | Aditya Ruhela | 2022DA04599 | 100% |
| 3 | Manikandan S | 2022DA04396 | 100% |
| 4 | Sheikh Rizwan | NA | 0% |

**Problem 1**

The PEAS (Performance measure, Environment, Actuator and Sensor) for the case are as follows-

Performance measure: The performance measure in the two-player game would be the number of points scored by each player.

Environment: The environment in the two-player game would be the Crossword puzzle board, where there are places at which the letters cannot be typed and other places where the word has to be placed.

Actuators: The actuators in a two-player game would be the words that are used to be placed at appropriate locations.

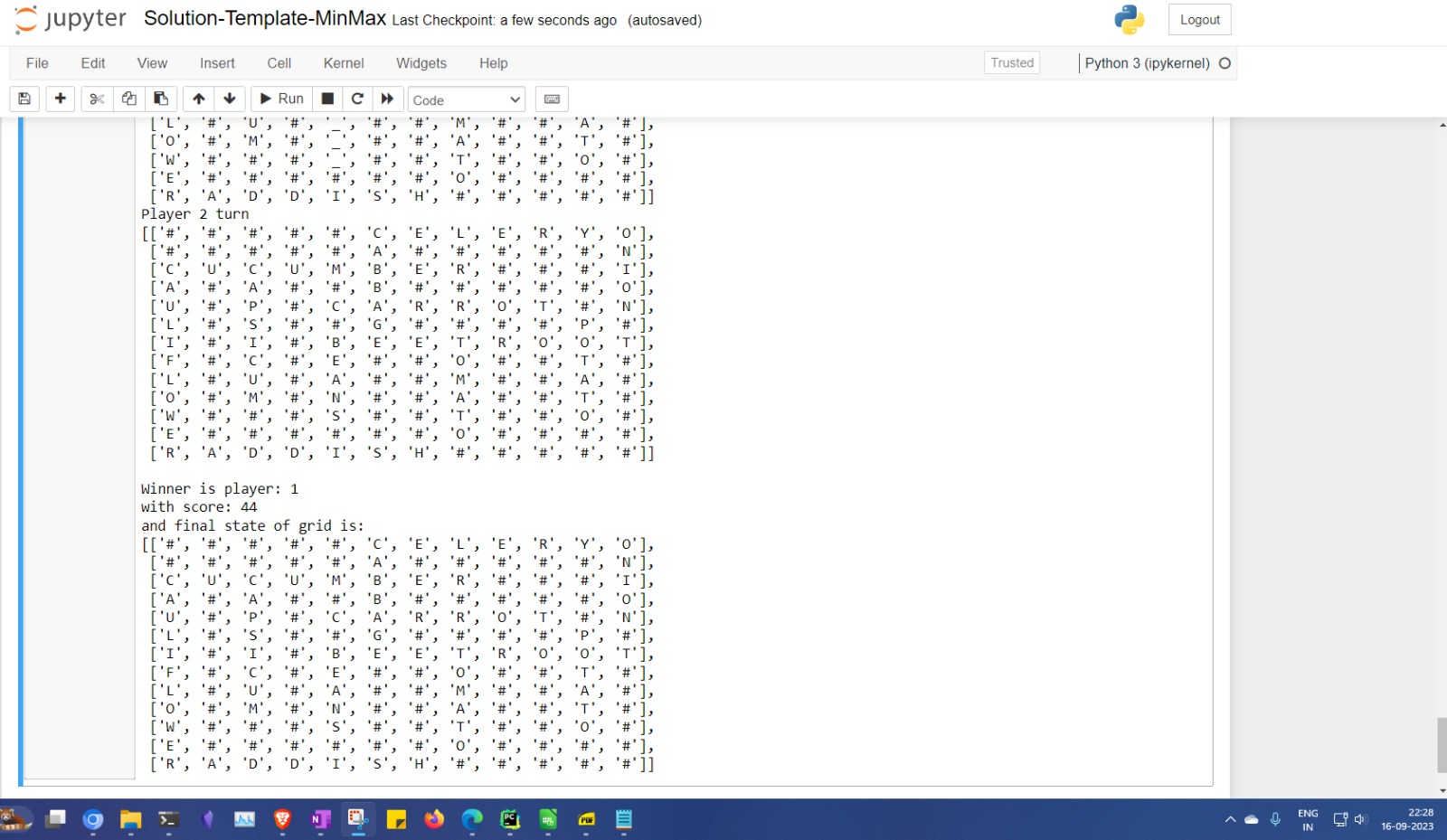
Sensors: The sensors in the two-player crossword game would be the physical or virtual components that allow players to receive information about the state of the game. This could include the sensor that identifies if the intersecting words have similar letter at the intersection.

In order to solve the crossword puzzle, the actuator first chooses the longest word and tries to fit it in to the possible places. Since there is only 1 place with length of 11, the longest word is placed there.

Subsequently, the program finds other words of less length.

If the position of the first word is correct, the second player places the next smaller length word is of 8 letters and there are 3 words of 8 letters. Hence the actuator places a word in all the possible places where the word can fit. Then it matches the intersecting words, if there are any. Based on the intersecting words, the pruning happens.

**Output with time and date stmp**

****

**Code and output**

from copy import deepcopy

PLAYER\_1 = 1

PLAYER\_2 = 2

fillable\_positions = None

def minimax(grid, player, depth=0, player1\_score=0, player2\_score=0):

if depth == len(words\_len): # means all words are explored

return player, player1\_score if player == PLAYER\_1 else player2\_score, grid

word\_to\_place, word\_len = words\_len[depth]

grid\_copy = deepcopy(grid)

if player == PLAYER\_1:

print('Player 1 turn')

for fillable\_possition in pick\_partially\_filled\_positions(grid, word\_len, fillable\_positions[word\_len]):

can\_word\_be\_place = check\_if\_word\_can\_be\_place(word\_to\_place, grid\_copy, fillable\_possition)

if can\_word\_be\_place:

write\_word\_to\_grid(word\_to\_place, grid\_copy, fillable\_possition)

pprint(grid\_copy)

fillable\_possition['filled'] = True

p, result\_score, result\_grid = minimax(grid\_copy, PLAYER\_2, depth + 1, player1\_score + word\_len,

player2\_score)

break

else:

p, result\_score, result\_grid = minimax(grid\_copy, PLAYER\_2, depth + 1, player1\_score - 1, player2\_score)

return PLAYER\_1, result\_score, result\_grid

else:

print('Player 2 turn')

for fillable\_possition in pick\_partially\_filled\_positions(grid, word\_len, fillable\_positions[word\_len]):

can\_word\_be\_place = check\_if\_word\_can\_be\_place(word\_to\_place, grid\_copy, fillable\_possition)

if can\_word\_be\_place:

write\_word\_to\_grid(word\_to\_place, grid\_copy, fillable\_possition)

pprint(grid\_copy)

fillable\_possition['filled'] = True

p, result\_score, result\_grid = minimax(grid\_copy, PLAYER\_1, depth + 1, player1\_score,

player2\_score + word\_len)

break

else:

p, result\_score, result\_grid = minimax(grid\_copy, PLAYER\_1, depth + 1, player1\_score, player2\_score - 1)

return PLAYER\_2, result\_score, result\_grid

def get\_empty\_place\_count\_by(grid,position):

From = position['from']

To = position['to']

cnt = 0

if From[0] == To[0]: #horizontal

for i in range(From[1],To[1]+1):

if grid[From[0]][i] == '\_': cnt+=1

else: #vertical

for i in range(From[0],To[0]+1):

if grid[i][From[1]] == '\_': cnt+=1

return cnt

def pick\_partially\_filled\_positions(grid, word\_len, some\_fillable\_positions):

partially\_filled\_places = []

#q = PriorityQueue()

for p in [fp for fp in fillable\_positions[word\_len] if not fp['filled']]:

partially\_filled\_places.append((get\_empty\_place\_count\_by(grid,p),p))

if len(partially\_filled\_places) == 0:return some\_fillable\_positions

pl,positions\_to\_return = zip(\*sorted(partially\_filled\_places,key=lambda x:x[0]))

return positions\_to\_return

def populate\_position(fillable\_cells, positions):

len\_word = len(fillable\_cells)

if len\_word >= MIN\_LEN:

postion = {'from': fillable\_cells[0], 'to': fillable\_cells[-1],'filled':False}

if len\_word in positions.keys():

positions[len\_word].append(postion)

else:

positions[len\_word] = [postion]

return []

def locate\_fillable\_possitions(grid):

# in the grid mark the fillable cell with '\_'

n\_rows = len(grid)

if n\_rows > 0:

n\_cols = len(grid[0])

else:

n\_cols = 0

positions = {}

# trace horizontally

for i in range(n\_rows):

fillable\_cells = []

for j in range(n\_cols + 1):

if j == n\_cols:

fillable\_cells = populate\_position(fillable\_cells, positions)

elif grid[i][j] == '\_':

fillable\_cells.append((i, j))

else:

fillable\_cells = populate\_position(fillable\_cells, positions)

# trace vertically

for i in range(n\_cols):

fillable\_cells = []

for j in range(n\_rows + 1):

try:

if j == n\_rows:

fillable\_cells = populate\_position(fillable\_cells, positions)

elif grid[j][i] == '\_':

fillable\_cells.append((j, i))

else:

fillable\_cells = populate\_position(fillable\_cells, positions)

except:

print(j, i)

return positions

def check\_if\_word\_can\_be\_place(word,grid,position):

From = position['from']

To = position['to']

j = 0

if From[0] == To[0]: #horizontal

for i in range(From[1],To[1]+1):

if grid[From[0]][i] == '\_':

j += 1

continue

if grid[From[0]][i] != word[j]: return False

j += 1

return True

else: #vertical

for i in range(From[0],To[0]+1):

if grid[i][From[1]] == '\_':

j += 1

continue

if grid[i][From[1]] != word[j]: return False

j += 1

return True

def write\_word\_to\_grid(word,grid,position):

From = position['from']

To = position['to']

j = 0

if From[0] == To[0]: #horizontal

for i in range(From[1],To[1]+1):

grid[From[0]][i] = word[j]

j+=1

else: #vertical

for i in range(From[0],To[0]+1):

grid[i][From[1]] = word[j]

j+=1

#Code block - Start the game

from pprint import pprint

words = '''CELERYO

CUCUMBER

CABBAGE

ONION

CARROT

CAPSICUM

CAULIFLOWER

RADDISH

TOMATO

BEETROOT

POTATO

BEANS'''.split('\n')

words\_len = sorted([(w,len(w)) for w in words],key = lambda x:x[1],reverse=True)

MAX\_LEN = words\_len[0][1]

MIN\_LEN = words\_len[-1][1]

grid\_str = '''#,#,#,#,#,\_,\_,\_,\_,\_,\_,\_

#,#,#,#,#,\_,#,#,#,#,#,\_

\_,\_,\_,\_,\_,\_,\_,\_,#,#,#,\_

\_,#,\_,#,#,\_,#,#,#,#,#,\_

\_,#,\_,#,\_,\_,\_,\_,\_,\_,#,\_

\_,#,\_,#,#,\_,#,#,#,#,\_,#

\_,#,\_,#,\_,\_,\_,\_,\_,\_,\_,\_

\_,#,\_,#,\_,#,#,\_,#,#,\_,#

\_,#,\_,#,\_,#,#,\_,#,#,\_,#

\_,#,\_,#,\_,#,#,\_,#,#,\_,#

\_,#,#,#,\_,#,#,\_,#,#,\_,#

\_,#,#,#,#,#,#,\_,#,#,#,#

\_,\_,\_,\_,\_,\_,\_,#,#,#,#,#'''

grid = [line.split(',') for line in grid\_str.split('\n')]

fillable\_positions = locate\_fillable\_possitions(grid)

winner, score, final\_grid = minimax(grid,PLAYER\_1)

print('\nWinner is player:',winner,'\nwith score:',score,'\nand final state of grid is:')

pprint(final\_grid)

final output:

Player 1 turn

[['#', '#', '#', '#', '#', '\_', '\_', '\_', '\_', '\_', '\_', '\_'],

['#', '#', '#', '#', '#', '\_', '#', '#', '#', '#', '#', '\_'],

['C', '\_', '\_', '\_', '\_', '\_', '\_', '\_', '#', '#', '#', '\_'],

['A', '#', '\_', '#', '#', '\_', '#', '#', '#', '#', '#', '\_'],

['U', '#', '\_', '#', '\_', '\_', '\_', '\_', '\_', '\_', '#', '\_'],

['L', '#', '\_', '#', '#', '\_', '#', '#', '#', '#', '\_', '#'],

['I', '#', '\_', '#', '\_', '\_', '\_', '\_', '\_', '\_', '\_', '\_'],

['F', '#', '\_', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['L', '#', '\_', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['O', '#', '\_', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['W', '#', '#', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['E', '#', '#', '#', '#', '#', '#', '\_', '#', '#', '#', '#'],

['R', '\_', '\_', '\_', '\_', '\_', '\_', '#', '#', '#', '#', '#']]

Player 2 turn

[['#', '#', '#', '#', '#', '\_', '\_', '\_', '\_', '\_', '\_', '\_'],

['#', '#', '#', '#', '#', '\_', '#', '#', '#', '#', '#', '\_'],

['C', 'U', 'C', 'U', 'M', 'B', 'E', 'R', '#', '#', '#', '\_'],

['A', '#', '\_', '#', '#', '\_', '#', '#', '#', '#', '#', '\_'],

['U', '#', '\_', '#', '\_', '\_', '\_', '\_', '\_', '\_', '#', '\_'],

['L', '#', '\_', '#', '#', '\_', '#', '#', '#', '#', '\_', '#'],

['I', '#', '\_', '#', '\_', '\_', '\_', '\_', '\_', '\_', '\_', '\_'],

['F', '#', '\_', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['L', '#', '\_', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['O', '#', '\_', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['W', '#', '#', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['E', '#', '#', '#', '#', '#', '#', '\_', '#', '#', '#', '#'],

['R', '\_', '\_', '\_', '\_', '\_', '\_', '#', '#', '#', '#', '#']]

Player 1 turn

[['#', '#', '#', '#', '#', '\_', '\_', '\_', '\_', '\_', '\_', '\_'],

['#', '#', '#', '#', '#', '\_', '#', '#', '#', '#', '#', '\_'],

['C', 'U', 'C', 'U', 'M', 'B', 'E', 'R', '#', '#', '#', '\_'],

['A', '#', 'A', '#', '#', '\_', '#', '#', '#', '#', '#', '\_'],

['U', '#', 'P', '#', '\_', '\_', '\_', '\_', '\_', '\_', '#', '\_'],

['L', '#', 'S', '#', '#', '\_', '#', '#', '#', '#', '\_', '#'],

['I', '#', 'I', '#', '\_', '\_', '\_', '\_', '\_', '\_', '\_', '\_'],

['F', '#', 'C', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['L', '#', 'U', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['O', '#', 'M', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['W', '#', '#', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['E', '#', '#', '#', '#', '#', '#', '\_', '#', '#', '#', '#'],

['R', '\_', '\_', '\_', '\_', '\_', '\_', '#', '#', '#', '#', '#']]

Player 2 turn

[['#', '#', '#', '#', '#', '\_', '\_', '\_', '\_', '\_', '\_', '\_'],

['#', '#', '#', '#', '#', '\_', '#', '#', '#', '#', '#', '\_'],

['C', 'U', 'C', 'U', 'M', 'B', 'E', 'R', '#', '#', '#', '\_'],

['A', '#', 'A', '#', '#', '\_', '#', '#', '#', '#', '#', '\_'],

['U', '#', 'P', '#', '\_', '\_', '\_', '\_', '\_', '\_', '#', '\_'],

['L', '#', 'S', '#', '#', '\_', '#', '#', '#', '#', '\_', '#'],

['I', '#', 'I', '#', 'B', 'E', 'E', 'T', 'R', 'O', 'O', 'T'],

['F', '#', 'C', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['L', '#', 'U', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['O', '#', 'M', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['W', '#', '#', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['E', '#', '#', '#', '#', '#', '#', '\_', '#', '#', '#', '#'],

['R', '\_', '\_', '\_', '\_', '\_', '\_', '#', '#', '#', '#', '#']]

Player 1 turn

[['#', '#', '#', '#', '#', 'C', 'E', 'L', 'E', 'R', 'Y', 'O'],

['#', '#', '#', '#', '#', '\_', '#', '#', '#', '#', '#', '\_'],

['C', 'U', 'C', 'U', 'M', 'B', 'E', 'R', '#', '#', '#', '\_'],

['A', '#', 'A', '#', '#', '\_', '#', '#', '#', '#', '#', '\_'],

['U', '#', 'P', '#', '\_', '\_', '\_', '\_', '\_', '\_', '#', '\_'],

['L', '#', 'S', '#', '#', '\_', '#', '#', '#', '#', '\_', '#'],

['I', '#', 'I', '#', 'B', 'E', 'E', 'T', 'R', 'O', 'O', 'T'],

['F', '#', 'C', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['L', '#', 'U', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['O', '#', 'M', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['W', '#', '#', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['E', '#', '#', '#', '#', '#', '#', '\_', '#', '#', '#', '#'],

['R', '\_', '\_', '\_', '\_', '\_', '\_', '#', '#', '#', '#', '#']]

Player 2 turn

[['#', '#', '#', '#', '#', 'C', 'E', 'L', 'E', 'R', 'Y', 'O'],

['#', '#', '#', '#', '#', 'A', '#', '#', '#', '#', '#', '\_'],

['C', 'U', 'C', 'U', 'M', 'B', 'E', 'R', '#', '#', '#', '\_'],

['A', '#', 'A', '#', '#', 'B', '#', '#', '#', '#', '#', '\_'],

['U', '#', 'P', '#', '\_', 'A', '\_', '\_', '\_', '\_', '#', '\_'],

['L', '#', 'S', '#', '#', 'G', '#', '#', '#', '#', '\_', '#'],

['I', '#', 'I', '#', 'B', 'E', 'E', 'T', 'R', 'O', 'O', 'T'],

['F', '#', 'C', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['L', '#', 'U', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['O', '#', 'M', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['W', '#', '#', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['E', '#', '#', '#', '#', '#', '#', '\_', '#', '#', '#', '#'],

['R', '\_', '\_', '\_', '\_', '\_', '\_', '#', '#', '#', '#', '#']]

Player 1 turn

[['#', '#', '#', '#', '#', 'C', 'E', 'L', 'E', 'R', 'Y', 'O'],

['#', '#', '#', '#', '#', 'A', '#', '#', '#', '#', '#', '\_'],

['C', 'U', 'C', 'U', 'M', 'B', 'E', 'R', '#', '#', '#', '\_'],

['A', '#', 'A', '#', '#', 'B', '#', '#', '#', '#', '#', '\_'],

['U', '#', 'P', '#', '\_', 'A', '\_', '\_', '\_', '\_', '#', '\_'],

['L', '#', 'S', '#', '#', 'G', '#', '#', '#', '#', '\_', '#'],

['I', '#', 'I', '#', 'B', 'E', 'E', 'T', 'R', 'O', 'O', 'T'],

['F', '#', 'C', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['L', '#', 'U', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['O', '#', 'M', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['W', '#', '#', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['E', '#', '#', '#', '#', '#', '#', '\_', '#', '#', '#', '#'],

['R', 'A', 'D', 'D', 'I', 'S', 'H', '#', '#', '#', '#', '#']]

Player 2 turn

[['#', '#', '#', '#', '#', 'C', 'E', 'L', 'E', 'R', 'Y', 'O'],

['#', '#', '#', '#', '#', 'A', '#', '#', '#', '#', '#', '\_'],

['C', 'U', 'C', 'U', 'M', 'B', 'E', 'R', '#', '#', '#', '\_'],

['A', '#', 'A', '#', '#', 'B', '#', '#', '#', '#', '#', '\_'],

['U', '#', 'P', '#', 'C', 'A', 'R', 'R', 'O', 'T', '#', '\_'],

['L', '#', 'S', '#', '#', 'G', '#', '#', '#', '#', '\_', '#'],

['I', '#', 'I', '#', 'B', 'E', 'E', 'T', 'R', 'O', 'O', 'T'],

['F', '#', 'C', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['L', '#', 'U', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['O', '#', 'M', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['W', '#', '#', '#', '\_', '#', '#', '\_', '#', '#', '\_', '#'],

['E', '#', '#', '#', '#', '#', '#', '\_', '#', '#', '#', '#'],

['R', 'A', 'D', 'D', 'I', 'S', 'H', '#', '#', '#', '#', '#']]

Player 1 turn

[['#', '#', '#', '#', '#', 'C', 'E', 'L', 'E', 'R', 'Y', 'O'],

['#', '#', '#', '#', '#', 'A', '#', '#', '#', '#', '#', '\_'],

['C', 'U', 'C', 'U', 'M', 'B', 'E', 'R', '#', '#', '#', '\_'],

['A', '#', 'A', '#', '#', 'B', '#', '#', '#', '#', '#', '\_'],

['U', '#', 'P', '#', 'C', 'A', 'R', 'R', 'O', 'T', '#', '\_'],

['L', '#', 'S', '#', '#', 'G', '#', '#', '#', '#', '\_', '#'],

['I', '#', 'I', '#', 'B', 'E', 'E', 'T', 'R', 'O', 'O', 'T'],

['F', '#', 'C', '#', '\_', '#', '#', 'O', '#', '#', '\_', '#'],

['L', '#', 'U', '#', '\_', '#', '#', 'M', '#', '#', '\_', '#'],

['O', '#', 'M', '#', '\_', '#', '#', 'A', '#', '#', '\_', '#'],

['W', '#', '#', '#', '\_', '#', '#', 'T', '#', '#', '\_', '#'],

['E', '#', '#', '#', '#', '#', '#', 'O', '#', '#', '#', '#'],

['R', 'A', 'D', 'D', 'I', 'S', 'H', '#', '#', '#', '#', '#']]

Player 2 turn

[['#', '#', '#', '#', '#', 'C', 'E', 'L', 'E', 'R', 'Y', 'O'],

['#', '#', '#', '#', '#', 'A', '#', '#', '#', '#', '#', '\_'],

['C', 'U', 'C', 'U', 'M', 'B', 'E', 'R', '#', '#', '#', '\_'],

['A', '#', 'A', '#', '#', 'B', '#', '#', '#', '#', '#', '\_'],

['U', '#', 'P', '#', 'C', 'A', 'R', 'R', 'O', 'T', '#', '\_'],

['L', '#', 'S', '#', '#', 'G', '#', '#', '#', '#', 'P', '#'],

['I', '#', 'I', '#', 'B', 'E', 'E', 'T', 'R', 'O', 'O', 'T'],

['F', '#', 'C', '#', '\_', '#', '#', 'O', '#', '#', 'T', '#'],

['L', '#', 'U', '#', '\_', '#', '#', 'M', '#', '#', 'A', '#'],

['O', '#', 'M', '#', '\_', '#', '#', 'A', '#', '#', 'T', '#'],

['W', '#', '#', '#', '\_', '#', '#', 'T', '#', '#', 'O', '#'],

['E', '#', '#', '#', '#', '#', '#', 'O', '#', '#', '#', '#'],

['R', 'A', 'D', 'D', 'I', 'S', 'H', '#', '#', '#', '#', '#']]

Player 1 turn

[['#', '#', '#', '#', '#', 'C', 'E', 'L', 'E', 'R', 'Y', 'O'],

['#', '#', '#', '#', '#', 'A', '#', '#', '#', '#', '#', 'N'],

['C', 'U', 'C', 'U', 'M', 'B', 'E', 'R', '#', '#', '#', 'I'],

['A', '#', 'A', '#', '#', 'B', '#', '#', '#', '#', '#', 'O'],

['U', '#', 'P', '#', 'C', 'A', 'R', 'R', 'O', 'T', '#', 'N'],

['L', '#', 'S', '#', '#', 'G', '#', '#', '#', '#', 'P', '#'],

['I', '#', 'I', '#', 'B', 'E', 'E', 'T', 'R', 'O', 'O', 'T'],

['F', '#', 'C', '#', '\_', '#', '#', 'O', '#', '#', 'T', '#'],

['L', '#', 'U', '#', '\_', '#', '#', 'M', '#', '#', 'A', '#'],

['O', '#', 'M', '#', '\_', '#', '#', 'A', '#', '#', 'T', '#'],

['W', '#', '#', '#', '\_', '#', '#', 'T', '#', '#', 'O', '#'],

['E', '#', '#', '#', '#', '#', '#', 'O', '#', '#', '#', '#'],

['R', 'A', 'D', 'D', 'I', 'S', 'H', '#', '#', '#', '#', '#']]

Player 2 turn

[['#', '#', '#', '#', '#', 'C', 'E', 'L', 'E', 'R', 'Y', 'O'],

['#', '#', '#', '#', '#', 'A', '#', '#', '#', '#', '#', 'N'],

['C', 'U', 'C', 'U', 'M', 'B', 'E', 'R', '#', '#', '#', 'I'],

['A', '#', 'A', '#', '#', 'B', '#', '#', '#', '#', '#', 'O'],

['U', '#', 'P', '#', 'C', 'A', 'R', 'R', 'O', 'T', '#', 'N'],

['L', '#', 'S', '#', '#', 'G', '#', '#', '#', '#', 'P', '#'],

['I', '#', 'I', '#', 'B', 'E', 'E', 'T', 'R', 'O', 'O', 'T'],

['F', '#', 'C', '#', 'E', '#', '#', 'O', '#', '#', 'T', '#'],

['L', '#', 'U', '#', 'A', '#', '#', 'M', '#', '#', 'A', '#'],

['O', '#', 'M', '#', 'N', '#', '#', 'A', '#', '#', 'T', '#'],

['W', '#', '#', '#', 'S', '#', '#', 'T', '#', '#', 'O', '#'],

['E', '#', '#', '#', '#', '#', '#', 'O', '#', '#', '#', '#'],

['R', 'A', 'D', 'D', 'I', 'S', 'H', '#', '#', '#', '#', '#']]

Winner is player: 1

with score: 44

and final state of grid is:

[['#', '#', '#', '#', '#', 'C', 'E', 'L', 'E', 'R', 'Y', 'O'],

['#', '#', '#', '#', '#', 'A', '#', '#', '#', '#', '#', 'N'],

['C', 'U', 'C', 'U', 'M', 'B', 'E', 'R', '#', '#', '#', 'I'],

['A', '#', 'A', '#', '#', 'B', '#', '#', '#', '#', '#', 'O'],

['U', '#', 'P', '#', 'C', 'A', 'R', 'R', 'O', 'T', '#', 'N'],

['L', '#', 'S', '#', '#', 'G', '#', '#', '#', '#', 'P', '#'],

['I', '#', 'I', '#', 'B', 'E', 'E', 'T', 'R', 'O', 'O', 'T'],

['F', '#', 'C', '#', 'E', '#', '#', 'O', '#', '#', 'T', '#'],

['L', '#', 'U', '#', 'A', '#', '#', 'M', '#', '#', 'A', '#'],

['O', '#', 'M', '#', 'N', '#', '#', 'A', '#', '#', 'T', '#'],

['W', '#', '#', '#', 'S', '#', '#', 'T', '#', '#', 'O', '#'],

['E', '#', '#', '#', '#', '#', '#', 'O', '#', '#', '#', '#'],

['R', 'A', 'D', 'D', 'I', 'S', 'H', '#', '#', '#', '#', '#']]

**PROBLEM 2**

The PEAS (Performance measure, Environment, Actuator and Sensor) for the case are as follows-

* **Performance Measure**- The performance efficiency and effectiveness of the rover is measured on the basis of the ability to move in left, right, forward and backward directions. The rover should be able to complete its task in a timely and efficient manner.
* **Environment-** The environment of the rover would be the surroundings on the moon.
* **Actuators**- The actuators of the robot that rescues the rabbit are its movement capabilities. The rover can move in four directions, left, right, forward and backward. It uses these movements to navigate and find the optimal path to the finish position.
* **Sensors-** The rover has sensors that allows it to perceive the environment around it and helps it more in safe directions. The rover can use its sensors to detect obstacles. The sensor also allows the rover to determine its present position and helps it in moving towards the goal position.

**Explaination**

The rover would move in directions Left, right, forward and backward on the basis of the input provided by the user.

The C0, C1, C2 and C3 directions are based on rules built using given decision using the input control parameters.

**Decision Tree**

a3 = false

| a10 = false: c0 (11.0/1.0)

| a10 = true

| | a2 = false

| | | a13 = false: c3 (12.0)

| | | a13 = true

| | | | a0 = false: c0 (2.0)

| | | | a0 = true: c3 (2.0)

| | a2 = true: c0 (7.0/2.0)

a3 = true

| a0 = false

| | a8 = false: c1 (12.0)

| | a8 = true

| | | a10 = false: c0 (5.0)

| | | a10 = true: c1 (4.0)

| a0 = true

| | a10 = false

| | | a8 = false

| | | | a4 = false: c2 (6.0/1.0)

| | | | a4 = true: c3 (2.0)

| | | a8 = true: c0 (7.0)

| | a10 = true

| | | a5 = false: c1 (5.0)

| | | a5 = true

| | | | a6 = false: c0 (7.0/2.0)

| | | | a6 = true: c3 (3.0)

Prolog Rules: (hand written)

is\_accuator\_activated(a3, false),

is\_accuator\_activated(a10, false) :- move(rover, forward).

is\_accuator\_activated(a3, false),

is\_accuator\_activated(a10, true),

is\_accuator\_activated(a2, true) :- move(rover, forward).

is\_accuator\_activated(a3, false),

is\_accuator\_activated(a10, true),

is\_accuator\_activated(a2, false),

is\_accuator\_activated(a13, false) :- move(rover, backward).

is\_accuator\_activated(a3, false),

is\_accuator\_activated(a10, true),

is\_accuator\_activated(a2, false),

is\_accuator\_activated(a13, true),

is\_accuator\_activated(a0, false) :- move(rover, forward).

is\_accuator\_activated(a3, false),

is\_accuator\_activated(a10, true),

is\_accuator\_activated(a2, false),

is\_accuator\_activated(a13, true),

is\_accuator\_activated(a0, true) :- move(rover, backward).

is\_accuator\_activated(a3, true),

is\_accuator\_activated(a0, false),

is\_accuator\_activated(a8, false) :- move(rover, left).

is\_accuator\_activated(a3, true),

is\_accuator\_activated(a0, false),

is\_accuator\_activated(a8, true),

is\_accuator\_activated(a10, false) :- move(rover, forward).

is\_accuator\_activated(a3, true),

is\_accuator\_activated(a0, false),

is\_accuator\_activated(a8, true),

is\_accuator\_activated(a10, true) :- move(rover, left).

is\_accuator\_activated(a3, true),

is\_accuator\_activated(a0, true),

is\_accuator\_activated(a10, false),

is\_accuator\_activated(a8, false),

is\_accuator\_activated(a4, false):- move(rover, right).

is\_accuator\_activated(a3, true),

is\_accuator\_activated(a0, true),

is\_accuator\_activated(a10, false),

is\_accuator\_activated(a8, false),

is\_accuator\_activated(a4, true):- move(rover, backward).

is\_accuator\_activated(a3, true),

is\_accuator\_activated(a0, true),

is\_accuator\_activated(a10, false),

is\_accuator\_activated(a8, true) :- move(rover, forward).

is\_accuator\_activated(a3, true),

is\_accuator\_activated(a0, true),

is\_accuator\_activated(a10, true),

is\_accuator\_activated(a5, false) :- move(rover, left).

is\_accuator\_activated(a3, true),

is\_accuator\_activated(a0, true),

is\_accuator\_activated(a10, true),

is\_accuator\_activated(a5, true),

is\_accuator\_activated(a6, false):- move(rover, forward).

is\_accuator\_activated(a3, true),

is\_accuator\_activated(a0, true),

is\_accuator\_activated(a10, true),

is\_accuator\_activated(a5, true),

is\_accuator\_activated(a6, true) :- move(rover, backward).

'''

import pandas as pd

df = pd.read\_csv('second.csv')

decision\_tree\_for\_prolog = '''a3 = false

| a10 = false: c0 (11.0/1.0)

| a10 = true

| | a2 = false

| | | a13 = false: c3 (12.0)

| | | a13 = true

| | | | a0 = false: c0 (2.0)

| | | | a0 = true: c3 (2.0)

| | a2 = true: c0 (7.0/2.0)

a3 = true

| a0 = false

| | a8 = false: c1 (12.0)

| | a8 = true

| | | a10 = false: c0 (5.0)

| | | a10 = true: c1 (4.0)

| a0 = true

| | a10 = false

| | | a8 = false

| | | | a4 = false: c2 (6.0/1.0)

| | | | a4 = true: c3 (2.0)

| | | a8 = true: c0 (7.0)

| | a10 = true

| | | a5 = false: c1 (5.0)

| | | a5 = true

| | | | a6 = false: c0 (7.0/2.0)

| | | | a6 = true: c3 (3.0)'''

import re

prolog\_rules = []

pipe\_map = {}

for line in decision\_tree\_for\_prolog.split('\n'):

line = line.replace(' ', '')

pipe\_cnt = line.count('|')

if pipe\_cnt == 0:

pipe\_map[pipe\_cnt] = line.replace('t', 'T').replace('f', 'F').replace('=', '==')

if '|' in line and ':' not in line:

pipe\_map[pipe\_cnt] = re.findall(r"\|(.\*)", line)[0].replace('|', '').replace('t', 'T').replace('f',

'F').replace('=',

'==')

if ':' in line:

pipe\_map[pipe\_cnt] = re.findall(r"\|(.\*?)\:", line)[0].replace('|', '').replace('t', 'T').replace('f',

'F').replace(

'=', '==')

rule = ' and '.join(

[pipe\_map[i].replace('a', 'a[').replace('==', ']==').replace('Fa[', 'Fa') for i in range(pipe\_cnt + 1)])

action = re.findall(r"\:(.\*?)\(", line)[0]

rule = ' '.join(rule.split())

prolog\_rules.append((rule, action))

def process\_parameters(input\_params):

params = list(input\_params)

if len(params) != 14:

print("please input exactly 14 input parameters")

return ''

if set(list(input\_params)) != set(('T', 'F')):

print("please enter only in format of 'T' and 'F'")

return ''

a = [True if p == 'T' else False for p in input\_params]

for rule, action in prolog\_rules:

if eval(rule):

return action

n\_matches = 0

for i in range(df.shape[0]):

gen = process\_parameters(''.join(['T' if p == True else 'F' for p in df.iloc[i].values if p in (True, False)]))

# print('generated:',)

act = df.iloc[i]['class']

# print('actual :',act)

# print('-'\*10)

if gen == act: n\_matches += 1

print('our Prolog rules based predictor is accurate:',100\*round(n\_matches / df.shape[0],3),'% times')

directions = {'c0':'Forword','c1':'Left','c2':'Right','c3':'Backward'}

user\_wants\_to\_stop = False

if \_\_name\_\_ == '\_\_main\_\_':

while not user\_wants\_to\_stop:

input\_params = input('enter 14 parameters to move rover(like TTTFTFTFFFTFTF) or enter "STOP" to exit:')

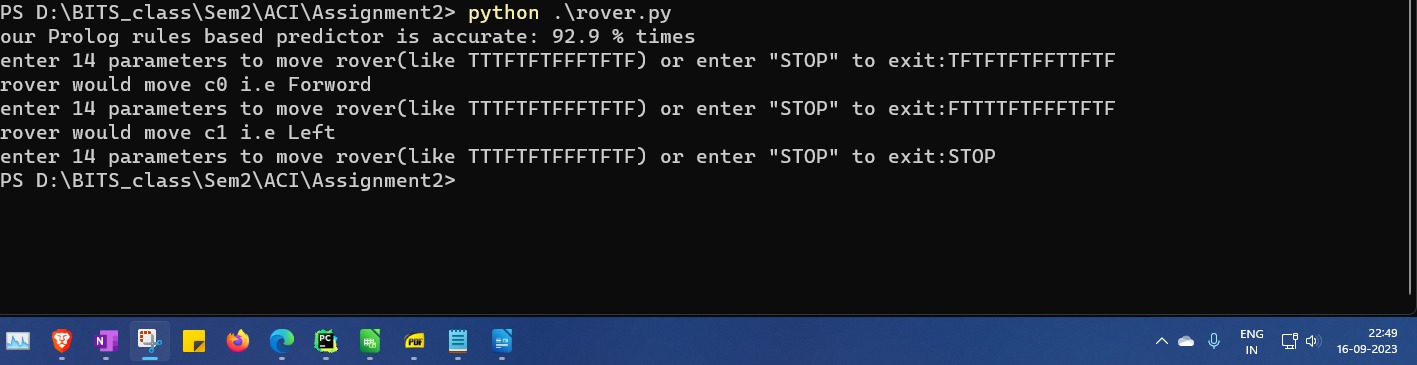
if input\_params == 'STOP' : break

output = process\_parameters(input\_params)

try:print('rover would move',output, 'i.e',directions[output])

except:pass

**Output**

****