AI Lab Record

Submitted By:

NAME - Arunabh Kalita

REGISTRATION NO. - RA1911003010375

SUBJECT NAME - Artificial Intelligence

SUBJECT CODE - 18CSC305J

BRANCH - Computer Science And Engineering

FACULTY NAME - Ms.M.Ranjani

Artificial Intelligence LAB-1

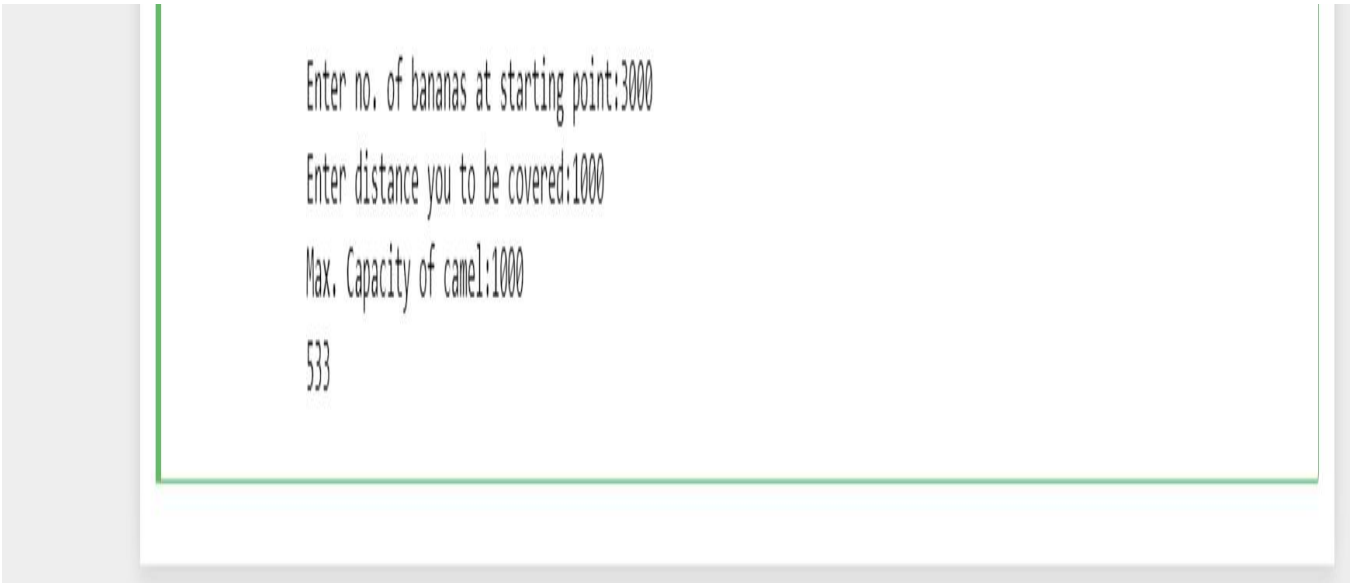
CAMEL BANANA PUZZLE

Date:13-1-22

Source Code:

```
total=int(input('Enter no. of bananas at starting point:'))
distance=int(input('Enter distance you to be covered:'))
max_capacity=int(input('Max. Capacity of camel:'))
lose=0
start_point=total
for i in range(distance):
    while start_point>0:
        start_point=start_point-max_capacity

        if start_point==1:
            lose=lose-1
        lose=lose+2

    lose=lose-1
    start_point=total-lose
    if start==0:
        break
print(start_point)
```

Output



Artificial Intelligence LAB-2

VACUUM CLEANER

Date:1-2-22

-Source Code:

```
import random

def display(room):
    print(room)

room = [
    [1, 1, 1, 1],
    [1, 1, 1, 1],
    [1, 1, 1, 1],
    [1, 1, 1, 1],
]
print("All the rooom are dirty")
display(room)

x =0
y= 0

while x < 4:
    while y < 4:
        room[x][y] = random.choice([0,1])
        y+=1
    x+=1
    y=0

print("Before cleaning the room I detect all of these random dirts")
display(room)
x =0
y= 0
z=0
while x < 4:
```

```
    while y < 4:
        if room[x][y] == 1:
            print("Vaccum in this location now,",x, y)
            room[x][y] = 0
            print("cleaned", x, y)
            z+=1
        y+=1
    x+=1
    y=0
pro= (100-((z/16)*100))
print("Room is clean now, Thanks for using the vacuum cleaner!")
display(room)
print('performance=',pro,'%')
```

Output

```
F:\College materials\Sem 6\AI\Practical\Lab2>python Lab2_VacuumCleaner.py
All the rooom are dirty
[[1, 1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 1]]
Before cleaning the room I detect all of these random dirts
[[0, 0, 1, 1], [1, 1, 1, 1], [1, 0, 0, 1], [1, 0, 1, 1]]
Vaccum in this location now, 0 2
cleaned 0 2
Vaccum in this location now, 0 3
cleaned 0 3
Vaccum in this location now, 1 0
cleaned 1 0
Vaccum in this location now, 1 1
cleaned 1 1
Vaccum in this location now, 1 2
cleaned 1 2
Vaccum in this location now, 1 3
cleaned 1 3
Vaccum in this location now, 2 0
cleaned 2 0
Vaccum in this location now, 2 3
cleaned 2 3
Vaccum in this location now, 3 0
cleaned 3 0
Vaccum in this location now, 3 2
cleaned 3 2
Vaccum in this location now, 3 3
cleaned 3 3
Room is clean now, Thanks for using the vacuum cleaner!
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
performance= 31.25 %
```

Artificial Intelligence LAB-3

N-QUEEN PROBLEM

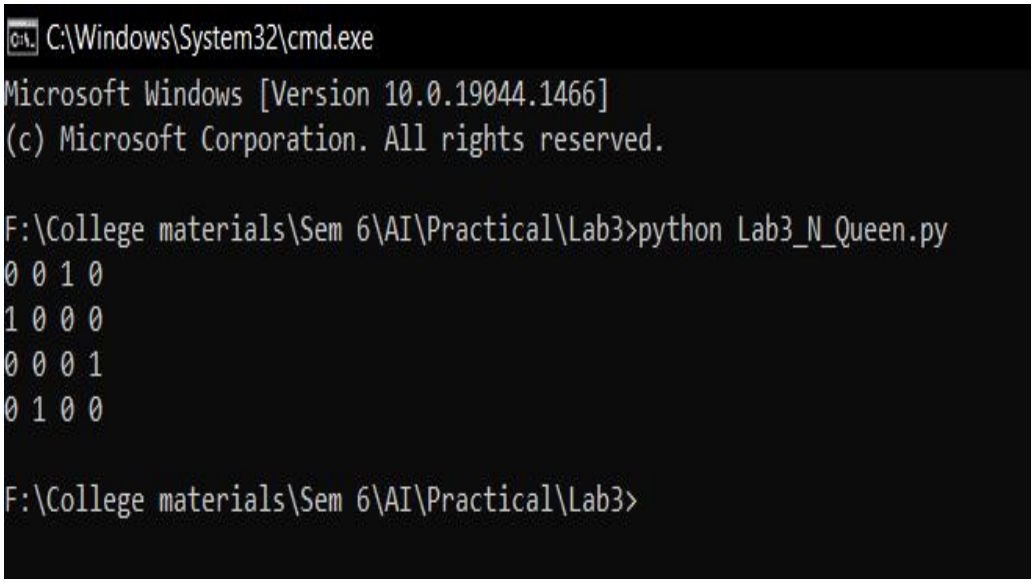
Date:8-2-22

-Source Code:

```
global N
N = 4

def printSolution(board):
    for i in range(N):
        for j in range(N):
            print (board[i][j], end = " ")
        print()


def isSafe(board, row, col):

    for i in range(col):
        if board[row][i] == 1:
            return False

    for i, j in zip(range(row, -1, -1),
                    range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    for i, j in zip(range(row, N, 1),
                    range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    return True

def solveNQUtil(board, col):
```

```
    if col >= N:
        return True

    for i in range(N):

        if isSafe(board, i, col):

            board[i][col] = 1

            if solveNQUtil(board, col + 1) == True:
                return True

            board[i][col] = 0

    return False

def solveNQ():
    board = [ [0, 0, 0, 0],
              [0, 0, 0, 0],
              [0, 0, 0, 0],
              [0, 0, 0, 0] ]

    if solveNQUtil(board, 0) == False:
        print ("Solution does not exist")
        return False

    printSolution(board)
    return True

solveNQ()
```

Output


C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19044.1466]
(c) Microsoft Corporation. All rights reserved.
F:\College materials\Sem 6\AI\Practical\Lab3>python Lab3_N_Queen.py
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
F:\College materials\Sem 6\AI\Practical\Lab3>

Artificial Intelligence LAB-4

N-QUEEN PROBLEM

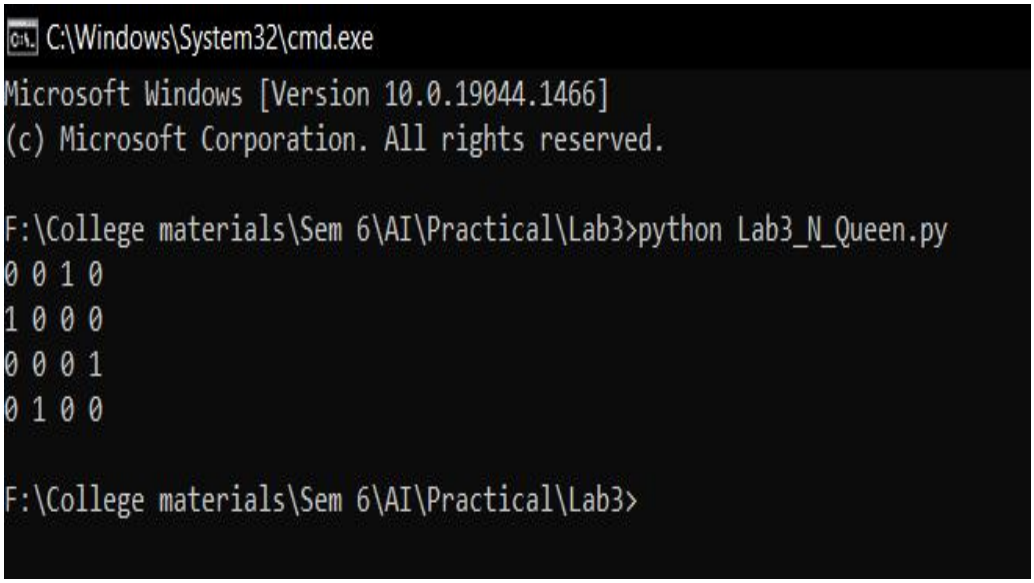
Date:8-2-22

-Source Code:

```
global N
N = 4

def printSolution(board):
    for i in range(N):
        for j in range(N):
            print (board[i][j], end = " ")
        print()


def isSafe(board, row, col):

    for i in range(col):
        if board[row][i] == 1:
            return False

    for i, j in zip(range(row, -1, -1),
                    range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    for i, j in zip(range(row, N, 1),
                    range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    return True

def solveNQUtil(board, col):
```

```
    if col >= N:
        return True

    for i in range(N):

        if isSafe(board, i, col):

            board[i][col] = 1

            if solveNQUtil(board, col + 1) == True:
                return True

            board[i][col] = 0

    return False

def solveNQ():
    board = [ [0, 0, 0, 0],
              [0, 0, 0, 0],
              [0, 0, 0, 0],
              [0, 0, 0, 0] ]

    if solveNQUtil(board, 0) == False:
        print ("Solution does not exist")
        return False

    printSolution(board)
    return True

solveNQ()
```

Output


C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19044.1466]
(c) Microsoft Corporation. All rights reserved.
F:\College materials\Sem 6\AI\Practical\Lab3>python Lab3_N_Queen.py
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
F:\College materials\Sem 6\AI\Practical\Lab3>

Artificial Intelligence LAB-5

A* Algorithm and Best First Search

Date:8-2-22

A* Algorithm

Solution:

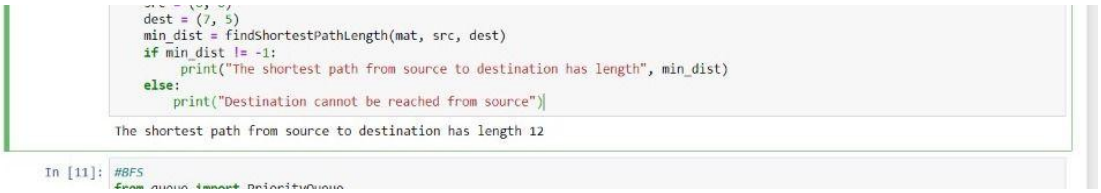
-Source Code:

```
import sys
def isSafe(mat, visited, x, y):
 return 0 <= x < len(mat) and 0 <= y < len(mat[0]) and \
 not (mat[x][y] == 0 or visited[x][y])
def findShortestPath(mat, visited, i, j, dest, min_dist=sys.maxsize, dist=0):

 if (i, j) == dest:
 return min(dist, min_dist)



 visited[i][j] = 1
if isSafe(mat, visited, i + 1, j):
 min_dist = findShortestPath(mat, visited, i + 1, j, dest, min_dist, dist + 1)

 if isSafe(mat, visited, i, j + 1):
 min_dist = findShortestPath(mat, visited, i, j + 1, dest, min_dist, dist + 1)



 if isSafe(mat, visited, i - 1, j):
 min_dist = findShortestPath(mat, visited, i - 1, j, dest, min_dist, dist + 1)

 if isSafe(mat, visited, i, j - 1):
 min_dist = findShortestPath(mat, visited, i, j - 1, dest, min_dist, dist + 1)

 visited[i][j] = 0

 return min_dist
```

```
def findShortestPathLength(mat, src, dest):

i, j = src

x, y = dest

if not mat or len(mat) == 0 or mat[i][j] == 0 or mat[x][y] == 0:
return -1

(M, N) = (len(mat), len(mat[0]))

visited = [[False for _ in range(N)] for _ in range(M)]
min_dist = findShortestPath(mat, visited, i, j, dest)
if min_dist != sys.maxsize:
return min_dist
else:
return -1
if __name__ == '__main__':
mat = [
[1, 1, 1, 1, 1, 0, 0, 1, 1, 1],
[0, 1, 1, 1, 1, 1, 0, 1, 0, 1],
[0, 0, 1, 0, 1, 1, 1, 0, 0, 1],
[1, 0, 1, 1, 1, 0, 1, 1, 0, 1],
[0, 0, 0, 1, 0, 0, 0, 1, 0, 1],
[1, 0, 1, 1, 1, 0, 0, 1, 1, 0],
[0, 0, 0, 0, 1, 0, 0, 1, 0, 1],
[0, 1, 1, 1, 1, 1, 1, 1, 0, 0],
[1, 1, 1, 1, 1, 0, 0, 1, 1, 1],
[0, 0, 1, 0, 0, 1, 1, 0, 0, 1]
]

src = (0, 0)
dest = (7, 5)
min_dist = findShortestPathLength(mat, src, dest)
if min_dist != -1:
```

print("The shortest path from source to destination has length", min_dist)
else:
print("Destination cannot be reached from source")

Output



Best First Search

Solution:

-Source Code:

```
from queue import PriorityQueue
v = 5
graph = [[] for i in range(v)]
def best_first_search(source, target, n):
 visited = [0] * n
 visited[0] = True
 pq = PriorityQueue()
 pq.put((0, source))
 while pq.empty() == False:
 u = pq.get()[1]
 print(u, end=" ")
 if u == target:
 break
 for v, c in graph[u]:
 if visited[v] == False:
 visited[v] = True
 pq.put((c, v))
 print()
def addedge(x, y, cost):
 graph[x].append((y, cost))
 graph[y].append((x, cost))
addedge(0, 1, 5)
addedge(0, 2, 1)
addedge(2, 3, 2)
addedge(1, 4, 1)
addedge(3, 4, 2)
source = 0
target = 4
best_first_search(source, target, v)
```

Output

```
addedge(3, 4, 2)
source = 0
target = 4
best_first_search(source, target, v)
```

```
0 2 3 4
```

Artificial Intelligence LAB-6

Min-Max Algorithms

Date:8-2-22

-Source Code:

```
from math import inf as infinity
from random import choice
import platform
import time
from os import system
HUMAN = -1
COMP = +1
board = [
 [0, 0, 0],
 [0, 0, 0],
 [0, 0, 0],
]
def evaluate(state):
 if wins(state, COMP):
 score = +1
 elif wins(state, HUMAN):

 score = -1
 else:
 score = 0
 return score
def wins(state, player):

 win_state = [
 [state[0][0], state[0][1], state[0][2]],
 [state[1][0], state[1][1], state[1][2]],
 [state[2][0], state[2][1], state[2][2]],
 [state[0][0], state[1][0], state[2][0]],
 [state[0][1], state[1][1], state[2][1]],
 [state[0][2], state[1][2], state[2][2]],
```

```
 [state[0][0], state[1][1], state[2][2]],
 [state[2][0], state[1][1], state[0][2]],
 ]
 if [player, player, player] in win_state:
 return True
 else:
 return False
def game_over(state):

 return wins(state, HUMAN) or wins(state, COMP)
def empty_cells(state):

 cells = []
 for x, row in enumerate(state):
 for y, cell in enumerate(row):
 if cell == 0:
 cells.append([x, y])
 return cells
def valid_move(x, y):

 if [x, y] in empty_cells(board):
 return True
 else:
 return False
def set_move(x, y, player):

 if valid_move(x, y):
 board[x][y] = player
 return True
 else:
 return False
def minimax(state, depth, player):

 if player == COMP:
 best = [-1, -1, -infinity]
```

```
 else:
 best = [-1, -1, +infinity]
 if depth == 0 or game_over(state):
 score = evaluate(state)
 return [-1, -1, score]
 for cell in empty_cells(state):
 x, y = cell[0], cell[1]
 state[x][y] = player
 score = minimax(state, depth - 1, -player)
 state[x][y] = 0
 score[0], score[1] = x, y
 if player == COMP:
 if score[2] > best[2]:
 best = score # max value
 else:
 if score[2] < best[2]:
 best = score # min value
 return best
def clean():
 """
 Clears the console
 """
 os_name = platform.system().lower()
 if 'windows' in os_name:
 system('cls')
 else:
 system('clear')
def render(state, c_choice, h_choice):
 """
 Print the board on console
 :param state: current state of the board
 """
 chars = {
 -1: h_choice,
 +1: c_choice,
```

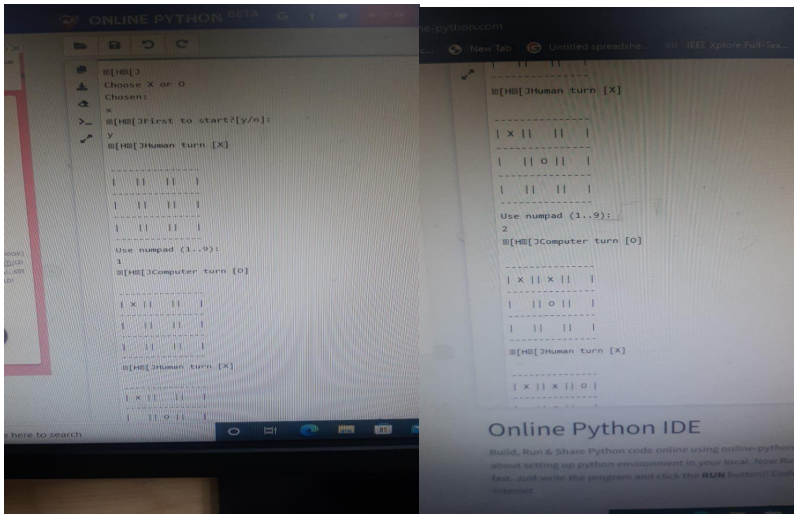
```
 0: ' '
 }
 str_line = '---------------'
 print('\n' + str_line)
 for row in state:
 for cell in row:
 symbol = chars[cell]
 print(f'| {symbol} |', end='')
 print('\n' + str_line)
def ai_turn(c_choice, h_choice):
 """
 It calls the minimax function if the depth < 9,
 else it choices a random coordinate.
 :param c_choice: computer's choice X or O
 :param h_choice: human's choice X or O
 :return:
 """
 depth = len(empty_cells(board))
 if depth == 0 or game_over(board):
 return
 clean()
 print(f'Computer turn [{c_choice}]')
 render(board, c_choice, h_choice)
 if depth == 9:
 x = choice([0, 1, 2])
 y = choice([0, 1, 2])
 else:
 move = minimax(board, depth, COMP)
 x, y = move[0], move[1]
 set_move(x, y, COMP)
 time.sleep(1)
def human_turn(c_choice, h_choice):

 depth = len(empty_cells(board))
 if depth == 0 or game_over(board):
```

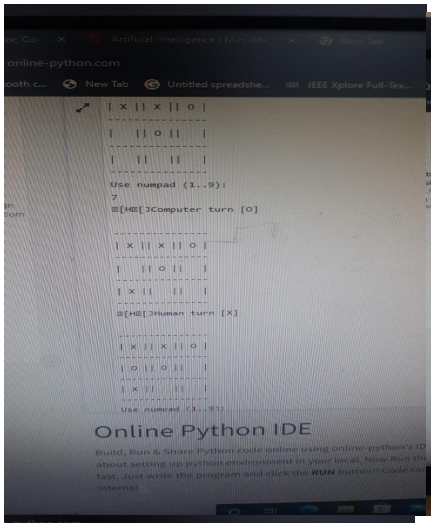
```
return
# Dictionary of valid moves
move = -1
moves = {
1: [0, 0], 2: [0, 1], 3: [0, 2],
4: [1, 0], 5: [1, 1], 6: [1, 2],
7: [2, 0], 8: [2, 1], 9: [2, 2],
}
clean()
print(f'Human turn [{h_choice}]')
render(board, c_choice, h_choice)
while move < 1 or move > 9:
try:
move = int(input('Use numpad (1..9): '))
coord = moves[move]
can_move = set_move(coord[0], coord[1], HUMAN)
if not can_move:
print('Bad move')
move = -1
except (EOFError, KeyboardInterrupt):
print('Bye')
exit()
except (KeyError, ValueError):
print('Bad choice')
def main():
"""
Main function that calls all functions
"""
clean()
h_choice = '' # X or O
c_choice = '' # X or O
first = '' # if human is the first
# Human chooses X or O to play
while h_choice != 'O' and h_choice != 'X':
try:
```

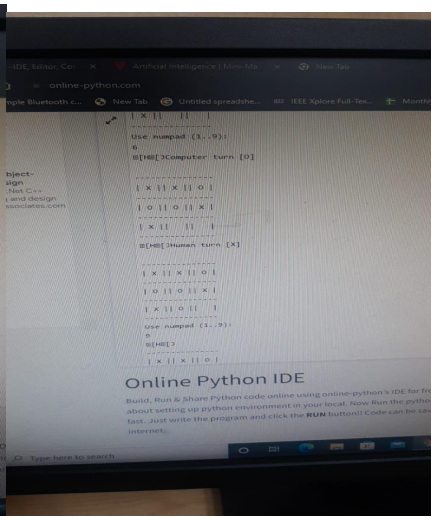
```
print('')
h_choice = input('Choose X or O\nChosen: ').upper()
except (EOFError, KeyboardInterrupt):
print('Bye')
exit()
except (KeyError, ValueError):
print('Bad choice')
# Setting computer's choice
if h_choice == 'X':
c_choice = 'O'
else:
c_choice = 'X'

# Human may starts first
clean()
while first != 'Y' and first != 'N':
try:
first = input('First to start?[y/n]: ').upper()
except (EOFError, KeyboardInterrupt):
print('Bye')
exit()
except (KeyError, ValueError):
print('Bad choice')
# Main loop of this game
while len(empty_cells(board)) > 0 and not game_over(board):
if first == 'N':
ai_turn(c_choice, h_choice)
first = ''
human_turn(c_choice, h_choice)
ai_turn(c_choice, h_choice)
# Game over message
if wins(board, HUMAN):
clean()
print(f'Human turn [{h_choice}]')
render(board, c_choice, h_choice)
```

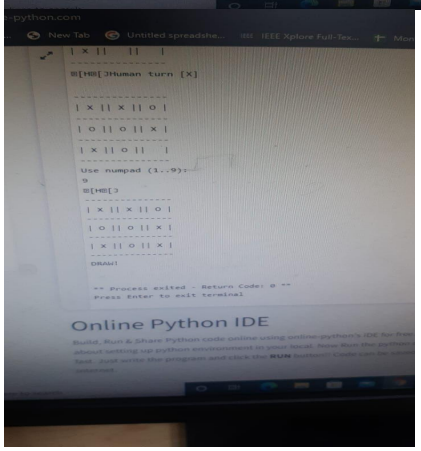
```
 print('YOU WIN!')
 elif wins(board, COMP):
 clean()
 print(f'Computer turn [{c_choice}]')
 render(board, c_choice, h_choice)
 print('YOU LOSE!')
 else:
 clean()
 render(board, c_choice, h_choice)
 print('DRAW!')
 exit()
if _name_ == '_main_':
 main()
```

Output




Use numpad (1..9):
7
Computer turn [O]
Human turn [X]
Online Python IDE


Use numpad (1..9):
6
Computer turn [O]
Human turn [X]
Online Python IDE


Human turn [X]
Use numpad (1..9):
9
DRAW!
** Process exited - Return Code: 0 **
Press Enter to exit terminal
Online Python IDE

Artificial Intelligence LAB-7

Unification and Resolution

Date:8-2-22

-Source Code:

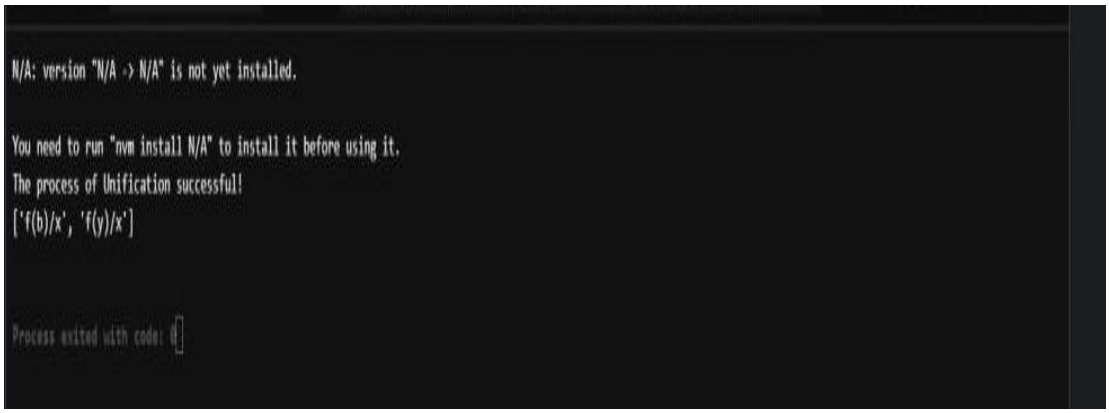
```
def get_index_comma(string): index_list =  list() par_count = 0

for i in range(len(string)):
if string[i] == ',' and par_count == 0: index_list.append(i)
elif string[i] == '(': par_count += 1 elif string[i] == ')':
par_count -= 1


return index_list



def is_variable(expr): for i in expr:
if i == '(' or i == ')': return False

return True



def process_expression(expr): expr = expr.replace(' ', '') index = None
for i in range(len(expr)): if expr[i] == '(':
index = i break
predicate_symbol = expr[:index]
expr = expr.replace(predicate_symbol, '') expr = expr[1:len(expr) - 1]
arg_list = list()
indices = get_index_comma(expr)


if len(indices) == 0: arg_list.append(expr)
else:
```

```
arg_list.append(expr[:indices[0]]) for i, j in zip(indices, indices[1:]):
arg_list.append(expr[i + 1:j])
arg_list.append(expr[indices[len(indices) - 1] + 1:])


return predicate_symbol, arg_list



def get_arg_list(expr):
_, arg_list = process_expression(expr)

flag = True while flag:
flag = False


for i in arg_list:
if not is_variable(i): flag = True
_, tmp = process_expression(i) for j in tmp:
if j not in arg_list: arg_list.append(j)
arg_list.remove(i)


return arg_list



def check_occurs(var, expr): arg_list = get_arg_list(expr) if var in arg_list:
return True


return False
```

```
def unify(expr1, expr2):


if is_variable(expr1) and is_variable(expr2): if expr1 == expr2:
return 'Null' else:

return False
elif is_variable(expr1) and not is_variable(expr2): if check_occurs(expr1,
expr2):
return False else:
tmp = str(expr2) + '/' + str(expr1) return tmp
elif not is_variable(expr1) and is_variable(expr2): if check_occurs(expr2,
expr1):
return False else:
tmp = str(expr1) + '/' + str(expr2) return tmp
else:
predicate_symbol_1, arg_list_1 = process_expression(expr1)
predicate_symbol_2, arg_list_2 = process_expression(expr2)

# Step 2
if predicate_symbol_1 != predicate_symbol_2: return False
# Step 3
elif len(arg_list_1) != len(arg_list_2): return False
else:
# Step 4: Create substitution list sub_list = list()

# Step 5:
for i in range(len(arg_list_1)):
tmp = unify(arg_list_1[i], arg_list_2[i])


if not tmp: return False
elif tmp == 'Null': pass
else:
if type(tmp) == list: for j in tmp:
```

```
sub_list.append(j)
else:
sub_list.append(tmp)


# Step 6 return sub_list



if     name     == ' main ':


f1 = 'Q(a, g(x, a), f(y))'
f2 = 'Q(a, g(f(b), a), x)' # f1 = input('f1 : ')
# f2 = input('f2 : ')


result = unify(f1, f2) if not result:
print('The process of Unification failed!') else:
print('The process of Unification successful!') print(result)
```

Output


N/A: version "N/A -> N/A" is not yet installed.
You need to run "nvm install N/A" to install it before using it.
The process of Unification successful!
['f(b)/x', 'f(y)/x']
Process exited with code: 0

Artificial Intelligence LAB-8

Knowledge Representation

Date:8-2-22

-Source Code:

```
go :- hypothesize(Animal),

write('I guess that the animal is: '), write(Animal),
nl, undo.
hypothesize(cheetah) :- cheetah, !. hypothesize(tiger)  :- tiger, !.
hypothesize(giraffe) :- giraffe, !. hypothesize(zebra)          :- zebra, !.
hypothesize(ostrich) :- ostrich, !. hypothesize(penguin) :- penguin, !.
hypothesize(albatross) :- albatross, !. hypothesize(unknown).
cheetah :- mammal, carnivore,
verify(has_tawny_color), verify(has_dark_spots).
tiger :- mammal,

carnivore, verify(has_tawny_color), verify(has_black_stripes).
giraffe :- ungulate,

verify(has_long_neck), verify(has_long_legs).
zebra :- ungulate, verify(has_black_stripes).


ostrich :- bird,

verify(does_not_fly), verify(has_long_neck).
penguin :- bird,

verify(does_not_fly), verify(swims), verify(is_black_and_white).
albatross :- bird,

verify(appears_in_story_Ancient_Mariner), verify(flys_well).
mammal :- verify(has_hair), !. mammal :- verify(gives_milk). bird  :-
verify(has_feathers), !. bird          :- verify(flys),
```

```
verify(lays_eggs). carnivore :- verify(eats_meat), !.
carnivore :- verify(has_pointed_teeth), verify(has_claws),
verify(has_forward_eyes).
ungulate :- mammal,

verify(has_hooves), !. ungulate :- mammal,
verify(chews_cud). ask(Question) :-
write('Does the animal have the following attribute: '), write(Question),
write('? '), read(Response),

nl,

( (Response == yes ; Response == y)

->

assert(yes(Question)) ; assert(no(Question)), fail).



:- dynamic yes/1,no/1. verify(S) :-
(yes(S)

->

true ; (no(S)
->

fail ; ask(S))).
undo :- retract(yes(_)),fail. undo :- retract(no(_)),fail. undo.
```

Output

```
?- go.
Does the animal have the following attribute: has_hair? n.

Does the animal have the following attribute: gives_milk? |: y.

Does the animal have the following attribute: eats_meat? |: y.

Does the animal have the following attribute: has_tawny_color? |: n.

Does the animal have the following attribute: has_hooves? |: y.

Does the animal have the following attribute: has_long_neck? |: n.

Does the animal have the following attribute: has_black_stripes? |: n.

Does the animal have the following attribute: has_feathers? |: y.

Does the animal have the following attribute: does_not_fly? |: n.

Does the animal have the following attribute: appears_in_story_Ancient_Mariner? Does the animal
 have the following attribute: appears_in_story_Ancient_Mariner?y
|: .

Does the animal have the following attribute: flys_well? |: y.

I guess that the animal is: albatross
true.

?-
```

```
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?-
% c:/Users/Admin/Desktop/animal.pl.txt compiled 0.00 sec, 29 clauses
?-
|    go.
Does the animal have the following attribute: has_hair? y.

Does the animal have the following attribute: eats_meat? |: y.

Does the animal have the following attribute: has_tawny_color? |: n.

Does the animal have the following attribute: has_hooves? |: n.

Does the animal have the following attribute: chews_cud? |: y.

Does the animal have the following attribute: has_long_neck? |: y.

Does the animal have the following attribute: has_long_legs? |: y.

I guess that the animal is: giraffe
true.

?-
```