17/11/23

Algo :-

(1) Choose one random b/w 0 & X using
choice = random.choice (0, 1)
print the choice

(2) take input from the player 1 using
for i range (9):
~~print and ask.~~

(3) print and ask square in which player want to play
j = input ("square :") , k = input ("0 or X")
arr[j] = ~~choice~~ k

(4) store the value 0 or 1 in the square ~~that~~ player input in form of array of 1 to 9.

(5) Take input 4 times, 2 times for player one and 2 times for player two ~~and~~ without using any condition

(6) From 5th input start checking the condition in Horizontal, ~~on~~ vertical & diagonal using index like for
Horizontal = arr[1] == arr[2] == arr[3]
Vertical = arr[1] == arr[4] == arr[7]
Diagonal = arr[1] == arr[5] == arr[9]

(7) with every input from 5th check condition and if out of 8 conditions any one condition get true end program

(8) print the winner according to last input of player if it's 0 mean 0 or 1 mean X.

1. Program :-

```
q = [[' ' for _ in range(3)]
flag = 0
rchoice = random.choice[(0, 1)]
for i in range(9):
    if(choice == 0 || choice == 1)
    row = int(input(f"player{i%2}, enter row(0-2):"))
    col = int input(f"player{i%2}, enter column:"))

    if 0<=row<=2 and 0<=col<=2 and
      q[row][col] == ' ':
        break.

    else:
      print("invalid Move, Try again)
      print(" Enter a valid input")
      q[row][col] = Move

    if(i>=5)
      if((arr[1] == arr[2] == arr[3]) ||
        (arr[4] == arr[5] == arr[6]) ||
        (arr[7] == arr[8] == arr[9]) ||
        (arr[1] == arr[4] == arr[7]) ||
        (arr[2] == arr[5] == arr[8]) ||
        (arr[3] == arr[6] == arr[9]) ||
        (arr[1] == arr[5] == arr[9]) ||
        (arr[3] == arr[5] == arr[7])

      flag = 1
      break
```

```
print('  |     |  ')
print(' '+arr[1]+' | '+arr[2]+' | 'arr[3]
print('  |     |  ')
print(' '+arr[4]+' | '+arr[5]+' | 'arr[6]
print('  |     |  ')
print(' arr[7]+' | '+arr[8]+' | 'arr[9]


if flag==1:
  print(F" player {i} wins! ")
else:
  print(" it's draw ")
```

output

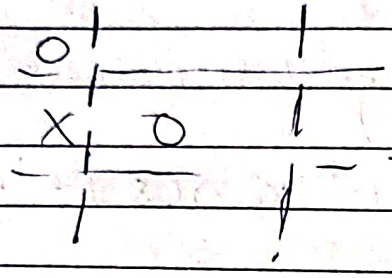
player 0
choose the row (0-2) : 0
choose the column (0-2) : 0

```
 0 |   |
---|---|---
   |   |
```

player 1
choose the row (0-2) : 0
choose the column (0-2) : 1

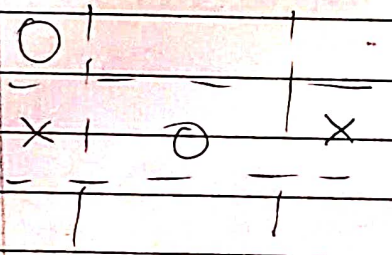
```
 0 | X |
---|---|---
   |   |
```

player 0
Choose the row (0-2) : 1
Choose the coloumn (0-2) : 1

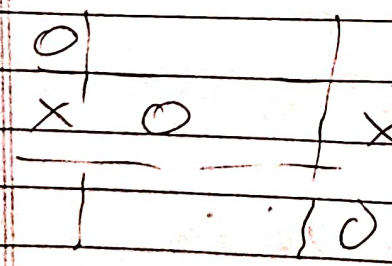


player 1

Choose the row (0-2) : 2
Choose the coloumn (0-2) : 1



player 0
Choose the row (0-2) : 2
Choose the coloumn (0-2) : 2



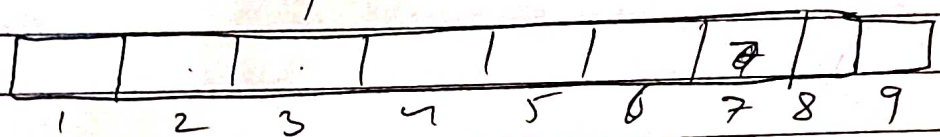
player 0 wins

18/11

2. 8-puzzle using BFS

Algorithm

1. Create a list of 9 element which represents 3x3 matrix with shuffled number from 1 to 8 and blank space with 0



2. The will be a shuffled list and the goal state where we need to reach.

3. To implement using the BFS there will be one queue to explore the grid in breadth-first manner.

4. The empty space will get swap with the number with it's (up ↑), (down ↓), (right →), (left ←) element

5. There is one visited queue to keep record of the visited state to avoid revisiting.

6. To check if the queue is empty or it reached it's goal state.

7. with each iteration of loop it will check if it's reached goal state or need to generate more successers of pattern.

Important

1. creating queue and and board.

```
def fboard(board)
    for r in board:
        print(r)
    print()
```

2. checking empty places

```
for i in range(3)
    for j in rang(3):
        if board[i][j] == 0;
```

3. To check if the move is valid.

```
if 0<=x<3 and 0<=y<=3
```

4. To swap the elements

```
if move == 'up'
    board[x2][y1],[x1][y2]
        = board[x2][y2],board[x1][y1]
```

5. queue traversal

```
q=queue()
```

6. get the blank position.
we can get the possible moves
move [0,1] = Right [0 -1] left
[1,0] = down [-1,0] - up

check if the move is valid.
and swap

if current_state == goal state.
~~return~~ return. path.

8-12-23

3. 8 puzzle using iterative deepening search algorithm.

① initialize the inital state and final state of 8 puzzle.

```
for i in range m:
    for j in range n:
        inital[i][i] = x;
```

Goal state.

```
goal state = [[1 2 3] [4 5 6] [7 8 0]]
```

② Find the blank space in the 8 puzzle by finding 0

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

③ write the possible action that can be applied on the 8 puzzle

```
if blank_row > 0:
    Move up
if blank_row < 2:
    Move down;
if blank_col > 0:
    Move left
if blank_col < 2;
    Move right
```

(4) Apply the action by swapping the position of all the possible action

if action = up
new_state[b_row][b_col]; new_state[b_row-1][b_col
= ~~{new_state[blank~~
= (new_state[b_row-1][b_col], new_state[b_row][b_col]

(5) Then check if the state is equal to the goal state.

goaltest(state, goal_state):
return state == goal_state

(6) Apply iterative deepening search algorithm
Run a while loop to trace the depth limit

(while i <= depth_limit)
{

}

(7) Apply the action on the puzzle to find possible written in step 3 & 4
node:
node = Puzzle(initial, node, action)

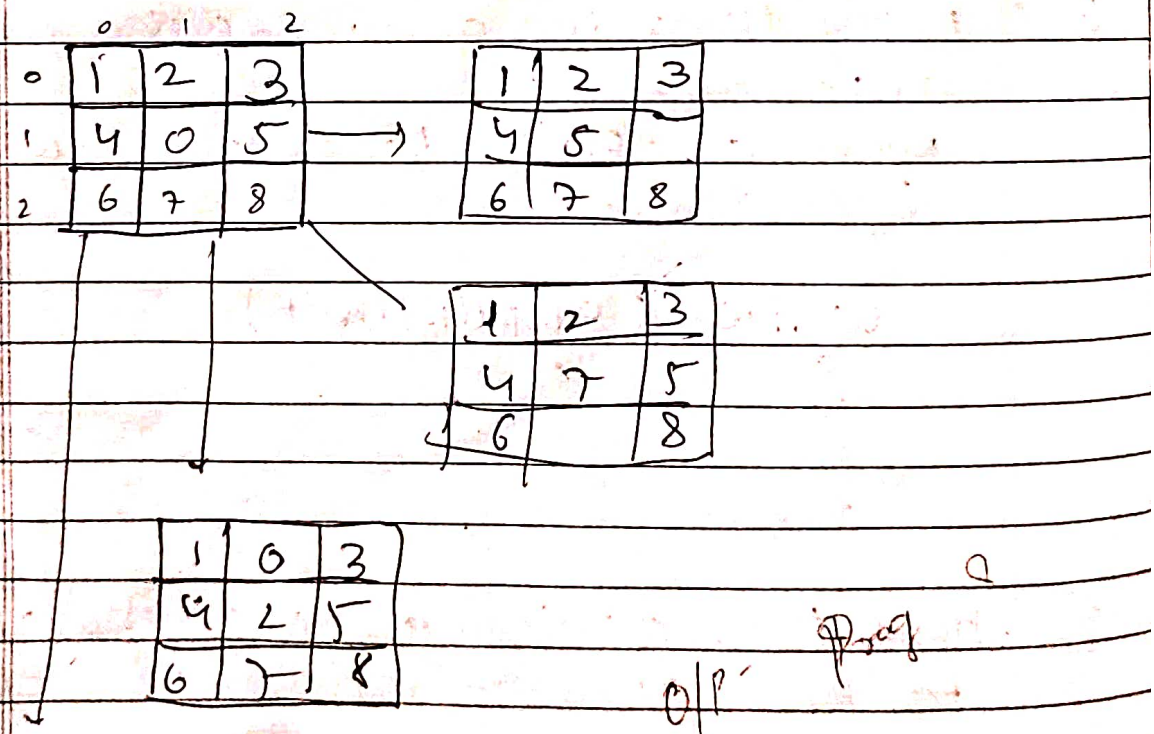
(8) Get the node & ~~dept~~ depth ~~and~~ and repeat the processes using recursion.

```
result = dls(node, depth-1);
```

(9) if the depth limit is 0 then the end the search there only

```
if depth.limit == 0;
    return "solved already";
```

if not then check goal state with every iteration of while loop



4 8 puzzle using A* Search.

(1) Take the Input of inital state state and the final state of 8 puzzle from user

```
initial = [int(i) for i in s.]
goal = [int(i) for i in s]
```

(2) Identify the empty tile position.

```
for i in initial:
    if i == 0 s
        pos = i;
        break;
```

(3) Find the row and column of the empty position

[1 2 3 4 0 5 6 7 8.]
0 1 2 3 4 5 6 7 8

```
row = pos // 3;
col = pos % 3;
```

(4) Apply action on the empty tile.

```
for move in [(0,1), (1,0) (0,-1) (-1,0)
```

(5) Check the move is valid or not

```
if 0<= new row < 3 for row
if 0<= new col < 3 for col.
```

(6) function to check if all the misplaced tile is placed

```
goal ( node, goal)
{ if (node = goal state)
    goal state = 1
}
```

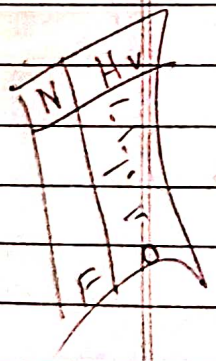
(7) Run a while loop till the goal state is reached or not

```
while ( goal state != 1)
```

and & to track the depth.

(8) Calculate f(n) by using f(n)=g(n)+h(n) where

g(n) = the value of the loop




```
h(n) = for i in node:
         if goal_state[i] != node[i]
           a++;
h(n) = a;
```

(9) compare f(n) at each depth and for the node which is min^m apply action step 4 & 5

```
result = (node, f(n))
```

8puzzle problem using IDDS

Code

```
def ids_dfs(puzzle, goal, strucure)
    input cu
    def dfs(route depth)
        if depth == 0
            return;
        if route = (-1) == goal
            return route
        for move in get Moves(while(')
            if Move not in route
                nexmove = dfs('not depth-1.)
        if henfle.
            return Rentroute
    for depth in iterturs_count()
        route = dfs(puzzle, depth)
        if route
            return route
def pasible Move (;stak);
    b = stak.in dn()
    d = []
    if b not in [0, 1];
        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');
```

```
per Move = [ ]
for i in d
        per-Move append (generate (state, i, y
return per Move

def generate (state. m, b)
    temp = State copy ( )
        if m == 'd' ;
    temp [b-3] / temp[b] = low

    return temp.

Uint : a+ [1, 2, 3, 0, 4, 6, 7, 8.]

goal state [1, 2, 3, 4, 5, 6, 7, 8, 0.]

route = id : def (unit:a, goal.posibile);

    If route:
        print ("success");
        printf ("path" : route)

else.
    print ("failed to find sol");
```

Output :-

success

path [[1 7 0 4 6 7 5) , (1, 2)]]

5 8 puzzle problem using A* algo

Code

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

def get_neighbours(state, moves)
    row, col = get_blank_pos(state)
    new_row = row + move[i][0]
    new_col = col + move[i][1]

    if 0<=new_row<3 and 0<=new_col<3:

        new_state = [link(row) for row in state]
        new_state[row][col];
        new_state[new_row]
            return new_state
    return none.

def is_goal(state)
    goal_state = [[1,2,3],[4,5,6],[7,8,9]]
        return state == goal_state.

def heuristic(state)
    distance = 0
        for i in range(3):
            for j in range(3):
```

```
if state[i][j] != 0
    goal = rows, goal_id
         = divmod[state(i)(j)]

    distance += abs(i-goal-0)

return distance;

def solve_puzzle(initial : state; now, move)
    open set [start_node.]
    closed_set = set()

while openset:
    current state, parent, new
        heap q, he appointne.)

    if goal(current_state)
        path()

while parent:
    path.insert(0, (move ; uni_state)
closed.set.add(temple (heap(tuple, curr state))

new ver = [(0, 1) ; (1,0), (0-1), (-1,0)

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

Setpath . = solve puzzle (initial-state)
```

```
if sol! path
    print ("move", move)
    for row in state;
        print(row)

print("---")
else
    print ("NO sol")
```

Output

```
1 2 3
4 5 6
7 8

1 2 3
4 5 6
0 7 8

1 2 3
4 5 6
7 0 8

4 2 3
4 5 6
7 8 0
```

29/12/23

29/12/23

Knowledge based Entailment

1. Define the function
   weather_entailment(hypo, premise, a, b, c)
   where a, b, c are predetermined condition

2. Check the entailment condition based on criteria.

~~if premise == '':~~
~~if a > 70~~
~~premise = '~~

if premise == "humid" and a > 70:
    return True

~~el~~if premise == "cloudy" and b > 50:
    return True

~~el~~if premise == "other_condition" and c
    return True

else
    return false

3. Example usage of the code.
premise condition = "humid" or
premise condition = "cloudy" or
premise condition = "other condition"

hypothesis_text = "The weather is uncomfortable".

4 Assume predetermined condition for humidity, cloudiness and other conditions like
humidity_condition = 75
cloudiness_condition = 60
other_condition = 40

5 call the function weather_entailment()

6 Check the result and print appropriate message.

Proved

Code

```
def weather_entailment(hypothesis, premise, a, b, c):
    if premise == "humidity" and a > 70:
        return True
    elif premise == "cloudy" and b > 80:
        return True
    elif premise == "other_condition" and c > 35:
        return True
    else:
        return false.

precond
premise_condition = "humid" or "cloudy"
                    or "other_condition"
hypothesis_text = "The weather is uncomfortable".

humidity_condition = 75.
Cloudness_condition = 80
other_condition = 30

result = weather_entailment(hypothesis_text
        premise_condition, humid_condition,
        cloudniess_condition, othercondition)

if result:
    printf("The hypothesis is entailed
        by {premise_condition} condition"
```

else

print("The hypothesis is ~~on~~ not entailed by the premise. condition) condition))

output

The hypothesis is is entailed by humid condition.
The weather is uncomfortable.

27/12/23

19/01/24

Unification

1. For input take two expression E_1 & E_2 that is needed to unify
2. Substitution σ - initially set to an empty substitution
3. The procedure to deal with cases

a) if E_1 is equal E_2, return the current substitution

```
if E1 == E2
    return sigma
```

b) if E_1 is a variable and E_1 is not equal to E_2 check if E_1 occurs in E_2 and if it does, return failure.

```
elif is_variable(E1):
    return unify_var(E1, E2, sigma)
```

c) if E_2 is a variable and E_2 is not equal to E_1 check if E_2 occurs in E_1 and if it does, return failure

```
elif is_variable(E2):
    return unify_var(E2, E1, sigma)
```

unify_var(var, exp, sigma)
{
if var in sigma:
return unify(sigma[var], exp, sigma)

elif exp_contains_var(expression, var):
return none

else
sigma[var] = expression.

}

d) If E_1 & E_2 are complex terms (non variable) with same functor
Recursively unify

elif is_complex(E_1) and is_complex(E_2)

4. if predicate(term1) ≠ predicate(term2)
return FAIL

5. Number of argument ≠ return fail

6. Set(SUBST) to NIL

7. for i = 1 to no. of element in term1
a) call unify(ith term1, ith term2)
put result into S
b) S = FAIL
result: Fail

c) if S ≠ NUL
a. Apply S to the remainder of L1 & L2
b. SUBST append (S SUBST)

8. Return SUBST

8 FOL to CNF conversion

1. Create a list of SKOLEM constants
2. 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 lower case and uppercase replace the uppercase attribte with a skolem function
3. replace <=> with '='
   transform as $\alpha \equiv (P \Rightarrow Q) \wedge (Q \Rightarrow P)$
4. replace => with '='
5. Apply demorgan's law
   replace ~[
   as ~P & ~Q if (was present)
   replace ~[
   as ~P | ~Q if (& was present)
   replace ∀ with ' '

Unification code

```
def is_variable(term):
    return is_instance(term, str) and
        term.is_lower()

def is_list(term):
    return is_instance(term, list)

def unify_var(var, expression, substitution)
    if (var in substitution:
        return unify(substitution[var],
            expression, substitution)

elif expression_contain_var(expression, var
    return None

else:
    substitution[var] = expression
    return substitution

def expression_contains_var(expression, var
    if expression == var:
        return True

    elif is_list(expression):
        return any(exp_contain_var(elem, var)
            for elem in expression)
    return False
```

```
def unify(expression1, expression2,
          substitution = None):

    if substitution is None:
        substitution = {}

    if expression1 == expression2:
        return substitution

    if is_variable(expression1):
        return unify_var(expression1, expression2,
                         substitution);

    elif is_variable(expression2):

        return unify_var(expression2, expression1,
                         substitution);

    elif is_list(expression1) and is_list(expression2):
        return unify_list(expression1, expression2,
                          substitution)

    return none

Expression1 = ['likes', 'John', 'pizza']
Expression2 = ['likes', 'x', 'pizza']

result = unify(expression1, expression2)
```

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 Skolemization(statement):
    SKOLEM_constants = [f'{chr(c)}' for
    c in range(ord('A'), ord('Z')+1)]

    matchs = re.findall("[∃].", statement)
    for match in matchs[::-1]:
        statement = statement.replace(match, "")
        for predicate in getpredicates(statement):
            attributes = getAttributes(predicate)

            if '' .join(attributes).islower():

    statement = statement.replace
    (match[1], SKOLEM_constants.pop(0))

    return statement.
```

```
def fol_to_cnf(fol):
    statement = fol.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

    return Skolemization(statement)
```

24/11/24

Incomplete program yet to execute.