```
01/10/2024
Lab-01
1.Program Title: Tic Tac Toe game
Code:
import random
def check_win(board, r, c):
  ch = 'X' if board[r - 1][c - 1] == 'X' else 'O'
  # Check the row
  if all(cell == ch for cell in board[r - 1]):
    return True
  # Check the column
  if all(board[i][c - 1] == ch for i in range(3)):
    return True
  # Check main diagonal
  if r == c and all(board[i][i] == ch for i in range(3)):
    return True
  # Check anti-diagonal
  if r + c == 4 and all(board[i][2 - i] == ch for i in range(3)):
    return True
  return False
def display_board(board):
  for row in board:
    print(" | ".join(row))
  print()
def find_block_move(board):
  # Check rows and columns for blocking opportunity
  for i in range(3):
    # Check rows
```

```
if board[i].count('X') == 2 and board[i].count('-') == 1:
       return i, board[i].index('-')
    # Check columns
    col = [board[0][i], board[1][i], board[2][i]]
    if col.count('X') == 2 and col.count('-') == 1:
       return col.index('-'), i
  # Check diagonals for blocking opportunity
  diag1 = [board[0][0], board[1][1], board[2][2]]
  if diag1.count('X') == 2 and diag1.count('-') == 1:
    idx = diag1.index('-')
    return idx, idx
  diag2 = [board[0][2], board[1][1], board[2][0]]
  if diag2.count('X') == 2 and diag2.count('-') == 1:
    idx = diag2.index('-')
    return idx, 2 - idx
  return None # No blocking move found
def find_winning_move(board):
  # Check rows and columns for winning opportunity
  for i in range(3):
    # Check rows
    if board[i].count('O') == 2 and board[i].count('-') == 1:
       return i, board[i].index('-')
    # Check columns
    col = [board[0][i], board[1][i], board[2][i]]
    if col.count('O') == 2 and col.count('-') == 1:
       return col.index('-'), i
  # Check diagonals for winning opportunity
  diag1 = [board[0][0], board[1][1], board[2][2]]
  if diag1.count('O') == 2 and diag1.count('-') == 1:
    idx = diag1.index('-')
```

```
return idx, idx
  diag2 = [board[0][2], board[1][1], board[2][0]]
  if diag2.count('O') == 2 and diag2.count('-') == 1:
    idx = diag2.index('-')
    return idx, 2 - idx
  return None # No winning move found
def bot_move(board):
  # First, check if there's a move to win
  winning_move = find_winning_move(board)
  if winning_move:
    r, c = winning_move
    board[r][c] = 'O'
    print(f"Bot placed O at winning position: ({r + 1}, {c + 1})")
    display_board(board)
    return r + 1, c + 1
  # Then, check if there's a move to block the human
  block_move = find_block_move(board)
  if block_move:
    r, c = block_move
    board[r][c] = 'O'
    print(f"Bot blocked X at position: ({r + 1}, {c + 1})")
    display_board(board)
    return r + 1, c + 1
  # Otherwise, make a random move
  available_moves = [(r, c) \text{ for } r \text{ in range}(3) \text{ for } c \text{ in range}(3) \text{ if board}[r][c] == '-']
  if available_moves:
    move = random.choice(available_moves)
    board[move[0]][move[1]] = 'O'
```

```
print(f"Bot placed O at position: ({move[0] + 1}, {move[1] + 1})")
    display_board(board)
    return move[0] + 1, move[1] + 1 # Return the move for win check
  return None, None
# Initial board setup
board = [['-', '-', '-'], ['-', '-', '-'], ['-', '-', '-']]
display_board(board)
xo = 1 # 1 for human, 0 for bot
flag = 0 # Flag to check for win or draw
while '-' in board[0] or '-' in board[1] or '-' in board[2]:
  if xo == 1: # Human's turn (X)
    print("Enter position to place X (row and column between 1-3):")
    x = int(input())
    y = int(input())
    if x > 3 or y > 3 or x < 1 or y < 1:
       print("Invalid position")
       continue
    if board[x - 1][y - 1] == '-':
       board[x - 1][y - 1] = 'X'
       xo = 0 # Switch to bot's turn
       display_board(board)
    else:
       print("Invalid position")
       continue
    if check_win(board, x, y):
       print("X wins!")
```

```
flag = 1
    break

else: # Bot's turn (O)
    print("Bot's turn:")
    x, y = bot_move(board)
    if x and y: # If bot made a valid move
        xo = 1 # Switch back to human's turn
        if check_win(board, x, y):
            print("O (Bot) wins!")
        flag = 1
            break

if flag == 0:
    print("Draw")

print("Game Over")
```

```
01/10/2024
program litle : tic tac toe gam
algori thm,
check-win (board, r, c):
   determine the letter placed ('x' (ori'o')
   stepz+
   check for a win in the row, column and diagonals return true if a win
   is found otherwise return false
 display-board (board):
   steply
   print the board
 find_block_move (boord):
   steply
   look for two 'x's in a row, column (or) diagonal with one empty space ('+')
   relum the blocking position (or) none
 bot_move (board):
                                          enter position to place a feebuc
   call find_block_move(board) if found place 'o' there
                 CE-1 a souted amotor bris
                                                enter position to place x
   step2+
   if no block is needed choose a random available move
  main algorithm
   steply
   initialize a 3x3 board with '-1
                                         bot blusted x of Postbon (33)
   set 'xo' (1 for player o for bot) and 'flag' (to check game status)
   while there are empty spots and on a wor) x ruly of noneof islas
   if Players turn (x):
   (i) prompt for row and column
   (ii) validate and place 'x' check for win
    if bots turn (0):
    i) call bot-move (board) check for win
                                  placed to at winning position (63)
   Step 41
   if no winner print 'draw'
   sters +
   Print 'game over'
```

```
output ,
                                                                            game over
                                                                            program little : voccum cleaner
                                                                           algorithma
  enter position to place x (now and column between 1-3)
                                                                           1. vaccum-cleaner-agent (perce
                                                                              input: a percept containing
                             denopole but amulas way
                                                                             Step1: extract location and
                                                                             step2!
                                                                                    if status is "pirty
                                                                                         Achon: Relvin
                                                                                    else is location is "f
  -1-1-
                                                                                        Action : Return
 bot's bin:
                                                                                    else if location is "B
 bot placed 0 at position (2,3)
                                   DOLE (16) ANYUJOS WOY D A) 2
                                                                                        Action! Return
  *1-1-
 -1-10
                                       snow the blocking position to an
                                                                                        Action: Return
 -1-1-
 enter position to place x (between
                                                                           2. main algorithm:
 enter position to place x (row and column between 1-3)
                                                                              step 1: initialize a list of
                               a nely boot to bood I was to I had
                                                                                      over time
                                                                              step 2 > create an empty 17
 X1-1-
    1 ×10
                                                                             steps : for each percept in
 -1-1
                                                                                         call vaccom-ch
bot's him:
                                                                                         append the ac
 bot blocked x at position (3,3)
                                                                                        corresponding .
                                                                             stepy: print the final pe
 es (ten plager oter bot) and 'flag' (to check gome status) o 1 x 1 -
                                                                           output ,
 -1-10
                                                                            percept : ['A', 'clean'], achi
enter Position to place x (row and column between 1-3)
                                                                            percept: ['A', 'dirly'], ach
                                                                            percept : ['B', 'clean']. achi
X1-1-
                                                                            percept = ['B', 'dirty'], actio
                                                                            percept : ['A', 'clean'], acho
x1x10
                                                                            percept : ['A', tclean'] , acti
-1-10
                                   are who donn's the
                                                                            percept sequence: [['A', 'cleo
bot's turn;
bot placed o at winning position (1,3)
                                                                                             ['A', '(le
                                                                            Oction sequence ; ['right',
*1-10
 XIXIO
-1-10
O(bot) wins
```

```
game over
                         program litte: voccom cleaner
                         algorithma
                         1. vaccum_cleaner_agent(percept);
                           input: a percept containing the current location and its status(Egs['A', 'Oirly'])
                          step1: extract location and status from the percept
                          Step2 :
                                   if status is "pirty":
                                        Action: Return "suck" (clean the current location)
                                  else if location is "A":
                                       Action: Return "right" (move to location B)
                                  else if location is "B"!
                                       Action: Return "left" (move to location A)
       con you tool
                                  else:
                                      Action: Return "No op" (this case should not occur in this simple work)
                         2. main algorithm:
   (binod) stom for
                           step 1: initialize a list of percepts representing the state of the environment
                                    over time
or bold bart los
                           step 2 > create an empty list to hold actions
                           steps: for each percept in the percept sequence:
                                        call vaccom-cleaner agent (percept) to determine the action
     millionto man
                                        append the action to the action list and print the percept and
             · (9013
                                        corresponding action
S EXE O SERIORIO
                           stepy! print the final percept and action sequences
019 (41) 68 178
                         output ,
                         percept: ['A', 'clean'], action: right
                          percept: ['A', 'dirty'], action: suck
                          percept : ['B', Iclean'], action : left
                          Percept = ['B', 'dirty'], action: such
                          percept: ['A', 'clean'], action: right
                          percept : ['A', t(lean'), action: right
                          percept sequence: [['A', 'clean'], ['A', 'dirty'], ['B', 'clean'], ['B', 'dirty'],
                                             ['A', '(lean'), ['A', clean')]
                          Oction sequence > ['right', 'suck', 'left', 'suck', 'right']
```

Output:

- | | -
- | | -
- | | -

```
Enter position to place X (row and column between 1-3):
1
1
X | - | -
- | - | -
- | - | -
Bot's turn:
Bot placed O at position: (3, 2)
X | - | -
- | - | -
- | 0 | -
Enter position to place X (row and column between 1-3):
2
2
X | - | -
- | X | -
- | 0 | -
Bot's turn:
Bot blocked X at position: (3, 3)
X | - | -
- | X | -
- | 0 | 0
Enter position to place X (row and column between 1-3):
3
1
X | - | -
```

```
- | X | - 

X | O | O

Bot's turn:

Bot blocked X at position: (2, 1)

X | - | - 
O | X | - 
X | O | O

Enter position to place X (row and column between 1-3): 1

3

X | - | X
O | X | - 
X | O | O

X wins!

Game Over
```

```
2.Program Title: vacuum cleaner
Code:
def vacuum_cleaner_agent(percept):
 A simple vacuum cleaner agent that operates in a two-location world.
 Args:
  percept: A list containing the current location and whether it is dirty.
       e.g., ['A', 'Dirty']
 Returns:
  The action to be taken by the agent (Left, Right, Suck, NoOp).
 location, status = percept
 if status == 'Dirty':
  return 'Suck'
 elif location == 'A':
  return 'Right'
 elif location == 'B':
  return 'Left'
 else:
  return 'NoOp' # Should not reach here in this simple world.
# Example percept sequence and action execution
percepts = [['A', 'Clean'], ['A', 'Dirty'], ['B', 'Clean'], ['B', 'Dirty'], ['A', 'Clean'], ['A', 'Clean']]
actions = []
for percept in percepts:
```

```
action = vacuum_cleaner_agent(percept)
actions.append(action)
print(f"Percept: {percept}, Action: {action}")
print("\nPercept Sequence:", percepts)
print("Action Sequence:", actions)
Algorithm:
```

```
game over
                        program little : voccum cleaner
                        algorithm
                        1. vaccum-cleaner-agent (percept):
                           input: a percept containing the current location and its status(egs('A', 'Oirly'))
                          step1: extract location and status from the percept
                          step2 !
                                  if Status is "pirty":
 od brood-polyst
                                       Action: Relvin "suct" (clean the current location)
 brood and lawy
                                  else if location is "A".
                                       Action: Return "right" (move to location B)
 NOW THE PARTY
                                 else if location is "B":
                                      Action: Return "left" (move to location A)
      Lost yet foot
                                 else :
            + 1.9355
                                      Action: Return "No op" (this case should not occur in this simple work)
                         2. main algorithm:
  (BELLEON) SHOWS TOUR
                           step 1: initialize a list of percepts representing the state of the environmens
                                    over time
of thold had the
                            step 2 : create an empty list to hold actions
                           step3: for each percept in the percept sequence:
                                        call vaccum-cleaner agent (percept) to determine the action
    malinagia man
                                        append the action to the action list and print the percept and
            413713
                                        corresponding action
                           stepy! Print the final percept and action sequences
            1 1512
 output ,
           41.4770
                         percept : ['A', 'clean'] , action : right
 HE WORL STORE
                          percept: ['A', 'dirty'], action: suck
                          percept : ['B', Iclean'], action : left
To not through
220 Mention 1
                          pellept = ['B', 'dirty'], action : such
                          percept: ['A', Iclean'], action : right
                          percept : ['A', releant], action : right
                          percept sequence: [('A', 'clean'), ('A', 'dirty'), ('B', 'clean'), ['B', 'dirty'],
                                             ['A', '(lean'), ['A', clean')]
                          action sequence : ['right', 'suck', 'left', 'suck', 'right']
```

Output:

Percept: ['A', 'Clean'], Action: Right

Percept: ['A', 'Dirty'], Action: Suck

Percept: ['B', 'Clean'], Action: Left

Percept: ['B', 'Dirty'], Action: Suck

Percept: ['A', 'Clean'], Action: Right

Percept: ['A', 'Clean'], Action: Right

Percept Sequence: [['A', 'Clean'], ['A', 'Dirty'], ['B', 'Clean'], ['B', 'Dirty'], ['A', 'Clean'], ['A', 'Clean']]

Action Sequence: ['Right', 'Suck', 'Left', 'Suck', 'Right', 'Right']

```
08/10/2024
Lab-02
3. Program title: 8 puzzle problem
Code:
import copy
# Directions for movement: up, down, left, right
moves = {'up': (-1, 0), 'down': (1, 0), 'left': (0, -1), 'right': (0, 1)}
# Check if a state is the goal state
def is_goal(state, goal_state):
  return state == goal_state
# Get the position of the empty space (0)
def get_empty_position(state):
  for i in range(3):
    for j in range(3):
      if state[i][j] == 0:
         return i, j
# Move the empty space in a specified direction if possible
def move_tile(state, direction):
  new_state = copy.deepcopy(state)
  empty_i, empty_j = get_empty_position(state)
  di, dj = moves[direction]
  new_i, new_j = empty_i + di, empty_j + dj
  if 0 <= new_i < 3 and 0 <= new_j < 3:
    new_state[empty_i][empty_j], new_state[new_i][new_j] = new_state[new_i][new_j],
new_state[empty_i][empty_j]
    return new_state
  return None
```

```
# Depth-limited search
def depth_limited_search(state, goal_state, depth_limit, path):
  if is_goal(state, goal_state):
    return state, path
  if depth_limit == 0:
    return None, []
  empty_i, empty_j = get_empty_position(state)
  for direction in moves:
    new_state = move_tile(state, direction)
    if new_state is not None and new_state not in path: # Avoid loops
       result, new_path = depth_limited_search(new_state, goal_state, depth_limit - 1, path +
[new_state])
       if result:
         return result, new_path
  return None, []
# Iterative deepening search
def iterative_deepening_search(initial_state, goal_state):
  depth = 0
  while True:
    result, path = depth_limited_search(initial_state, goal_state, depth, [initial_state])
    if result is not None:
       return path, depth
    depth += 1
# Print the state of the puzzle
def print_state(state):
```

```
for row in state:
    print(row)
  print()
# Test the 8-puzzle
initial_state = [
  [1, 2, 3],
  [4, 0, 5],
  [6, 7, 8]
]
goal_state = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 0]
]
# Solve the puzzle using iterative deepening search
solution_path, depth = iterative_deepening_search(initial_state, goal_state)
# Output the steps
print(f"Solution found in {depth} steps.\n")
print("Steps to reach the goal:")
for i, state in enumerate(solution_path):
  print(f"Step {i}:")
  print_state(state)
Algorithm:
```

4505/01/80 program litte: 8 puzzle problem algorithm , 1- is_goal(state, gaal_state): input a service State: current state of the puzzle must reduce two milesel trader goal-state: target configuration Process: check if state is equal to goal state we miss and a output: "A" if neithered it all return true if equal, otherwise return false else is location a "6". 2. get-empty_position(state): Actions Return "189" (more to localized a) input: state: current state of the puzzle action: Return "eso op" (this cose should not actor in this process: iterate through the 3x3 grid to find the position of o (empty space) return the coordinates (i, i) of the empty space 3 - move_file (state, direction): another Host of left glymp are super sayed sleps to each percept in the percept reprence. input: State: cument state of the puzzle direction: one of ('UP', down', 'left'; 'vight') Process: get him the final general and ortion requested create a deep copy of state get the position of the empty space calculate the new position based on direction if valid . swap the empty space with the adjacent tile return the new state if valid. Otherwise return none 4. depth_limited_search(state, goal_state, depth_limit, path); state, goal-state, depth-limit, path: corrent path taken if state is the goal return state and path if depth-limit is a relum none and an empty list

get the empty sp for each direction if the new - search reci if a solution output: return none and an 5. iterative - deeping - searce input: initial-state, goalblocere: initialize depth to o 100P: call depth_limited. if a solution is fou increment depth return the solution 6. Print_state (state): input : state: current state PAOCETI: print each yow of the output: display the corrent test 8 puzzle: 1-define 1- define inthial-state a 2- call iterative-deeping 3. Print the solution po

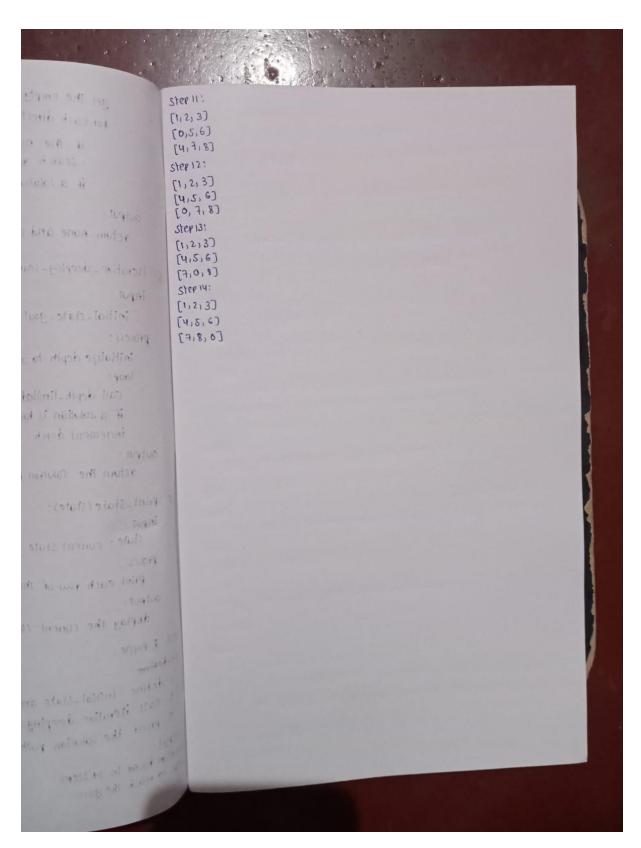
output.

Stepo:

solution found in 14 steps steps to reach the goal:

get the emply space position for each direction, attemp to move: if the new state is valid and not visited, call depth-limited - search recursively if a solution is found, return is output , THE RESERVE return none and an empty list if no solution is found al religion at 50/9 5. iterative - deeping - search (initial - state, goal - state): motor well a a bedieved of tells initial-state, goal-state. metale theiling STANOLETZ: initialize depth to o 100p: meina meiba call depth-limited-search with current depth tion of o (emply space) if a solution is found return the path and depth increment depth output: return the solution path and depth when found glans in stone : 1992 6. Print_State (state): sieps to each percept state: current state of the puzzle it'Il hargo print each row of the state I fould out total . History ! output: display the current configuration of the puzzle Dr. ["month . 'a'] : 195 R'Y test 8 puzzzle: adjacent tile 1-define 5 (51,1 , 97) 1- define initial-state and goal-state 2. Call iterative-deeping-search with # the initial-state and goal-state him none 3 Print the solution path and the number of steps taken to reach the goal mit, path); output. nt path taken solution found in 14 steps steps to reach the goal: Stepo: th . Pty list

[1,2,3]		
(20,4) 9	welling away glams out my	Step 11:
C FC 3. 97	the mention allement in more	[1,2,3]
de position and the	les estade to a kind affect a state war sin a	[0,5,6]
1 -1-1	their account	[4,7,8]
[1,2,3]		step 12:
[4,5,0]	if and it have it reduced it	[1,2,3]
[8,4,8]		[4,5,6]
Step 2:	formed of mandates are No. 1 to June 10 to 10	(0, 7, 8)
[1,2,3]	school and an emphy tist if no solution is found	ster 131
[4,5,8]	Winds to a state to the last	[4,5,6]
[6,7,0]	Cateth - book after the Heiling of the Book - Wheel	[7,0,1]
steps!	Lyn'	Step 14:
[1,2,3]	situal state, god- pale	[1,2,3]
(4,5,8)	- 21 CT - 12 CF - 31 CT2 - 10 (16)	[4,8,6)
2 [6,0,7]	THE PARTY NAMED IN COLUMN	[7,8,0]
Step 41	titualize depits to a	
[1,2,3]	Total Control of the	
[4,5,8]	digab teams seem assem bolimit digab tion	
[0,6,7]	state and other are more and if notices a	
steps:	isonemi desta	
[1,2,3]		
[0,5,8]	A November 1 and 1	
(4,6,7)	stan the Souther path and depth when found	
3 Stepe		
[1,2,3]	conficient states	
[4,6,7]		
Step 7:	lake contar trate of the purple	
[1,2,8]		
(5,6,8)	Per early you of the slate	
(4,0,7)	146	
step 8:		
[1,2,3]	here the continuity of the past	
[5,6,0]		
[4,7,8]		
518891		
[7,2,3]	statt took has statt tooks with	
[5,6,0]	and affects for the same of th	
4 [4, 7, 8]	ent makes open to wante and the makes of the sale one	
Step 10	of agent they to haunce off the special interpre-	
[1,2,3]	1,411	
[8,0,6]	could be at the control of the contr	
(4,7,8)		



Output:

Solution found in 14 steps.

Steps to reach the goal:

[4, 5, 0]			
[6, 7, 8]			
Step 2:			
[1, 2, 3]			
[4, 5, 8]			
[6, 7, 0]			
Step 3:			
[1, 2, 3]			
[4, 5, 8]			
[6, 0, 7]			
Step 4:			
[1, 2, 3]			
[4, 5, 8]			
[0, 6, 7]			
Step 5:			
[1, 2, 3]			
[0, 5, 8]			
[4, 6, 7]			
Step 6:			

Step 0:

[1, 2, 3]

[4, 0, 5]

[6, 7, 8]

Step 1:

[1, 2, 3]

[5, 0, 8]
[4, 6, 7]
Step 7:
[1, 2, 3]
[5, 6, 8]
[4, 0, 7]
Step 8:
[1, 2, 3]
[5, 6, 8]
[4, 7, 0]
Step 9:
[1, 2, 3]
[5, 6, 0]
[4, 7, 8]
Step 10:
[1, 2, 3]
[5, 0, 6]
[4, 7, 8]
Step 11:
[1, 2, 3]
[0, 5, 6]
[4, 7, 8]
Step 12:
[1, 2, 3]

[1, 2, 3]

- [4, 5, 6]
- [0, 7, 8]
- Step 13:
- [1, 2, 3]
- [4, 5, 6]
- [7, 0, 8]
- Step 14:
- [1, 2, 3]
- [4, 5, 6]
- [7, 8, 0]