1. Implement Tic-Tac-Toe Game

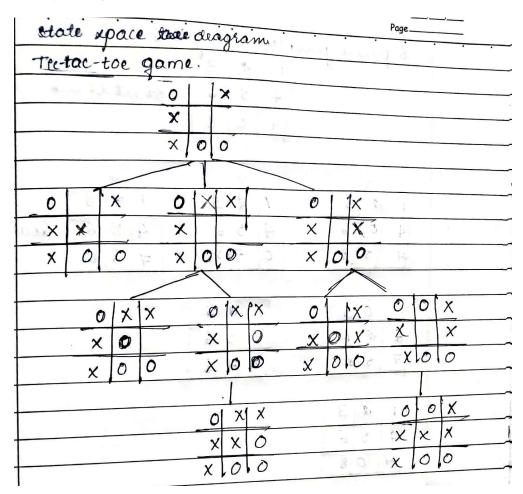
```
Code:
```

```
#Implementing tic-tac toe game
import math
# Board is represented as a list of 9 elements, initially empty
board = [''for _ in range(9)]
# Function to print the board
def print_board(board):
  for row in [board[i * 3:(i + 1) * 3] for i in range(3)]:
    print('| ' + ' | '.join(row) + ' |')
# Function to check if there is a winner
def winner(board, player):
  win conditions = [(0, 1, 2), (3, 4, 5), (6, 7, 8), (0, 3, 6), (1, 4, 7), (2, 5, 8), (0, 4, 8), (2, 4, 6)]
  for condition in win conditions:
    if all([board[i] == player for i in condition]):
       return True
  return False
# Function to check if the board is full
defis board full(board):
  return ' ' not in board
# Function to make a move
def make move(board, move, player):
  board[move] = player
# Minimax function to find the best move
def minimax(board, depth, is maximizing):
  if winner(board, 'O'): # AI wins
    return 1
  if winner(board, 'X'): # Player wins
    return-1
  if is board full(board): # Draw
    return 0
  if is maximizing:
    best score =-math.inf
    for i in range(9):
       if board[i] == ' ':
         board[i] = 'O'
         score = minimax(board, depth + 1, False)
         board[i] = ' '
         best_score = max(score, best_score)
    return best_score
  else:
    best score = math.inf
    for i in range(9):
       if board[i] == ' ':
         board[i] = 'X'
         score = minimax(board, depth + 1, True)
         board[i] = ' '
         best score = min(score, best score)
    return best_score
```

```
# Function to find the best move for the AI
def find best move(board):
  best_move = None
  best_score =-math.inf
  for i in range(9):
    if board[i] == ' ':
      board[i] = 'O'
      score = minimax(board, 0, False)
      board[i] = ' '
      if score > best_score:
        best_score = score
        best move = i
  return best move
# Main game loop
def play game():
  while True:
    print_board(board)
    # Player's move
    move = int(input("Enter your move (0-8): "))
    if board[move] != ' ':
      print("Invalid move. Try again.")
      continue
    make move(board, move, 'X')
    if winner(board, 'X'):
      print_board(board)
      print("Player wins!")
      break
    if is_board_full(board):
      print_board(board)
      print("It's a draw!")
      break
    # AI's move
    print("AI is making a move...")
    ai move = find best move(board)
    make_move(board, ai_move, 'O')
    if winner(board, 'O'):
      print_board(board)
      print("AI wins!")
      break
    if is_board_full(board):
      print_board(board)
      print("It's a draw!")
      break
play_game()
Output:
I I I I I
\perp
Enter your move (0-8): 1
Al is making a move...
| O | X | |
```

I + I + IEnter your move (0-8): 4 Al is making a move... | O | X | | | | X | | | |0| | Enter your move (0-8): 2 Al is making a move... | O | X | X | | | X | | |0|0|| Enter your move (0-8): 5 Al is making a move... | O | X | X | | O | X | X | |0|0| Al wins!

1.	Implementing tic-tac-toe game.
	Alongithm in a day parage with the
	1. Interaline the 3x3 boad vering liets in which
	a cell are emply miteally.
	d. Take the input from the user to thoose the
road!	Pacifican Systems of Appendix Systems
	8. If the position is nonempty excluded the today
8.8	" Ag Inplote most a famon" for all it to be to the
	4. level prize that , bridge south of the
: 16	3) while a function to display board to
	and current state of the board.
	4) would a function to check got wen, check all
e st fis	how, column and diagonal years
8 .	5) would a function to check for draw
	6) Define a function to accept the tours
	enput for human players more.
	If the cold is empty
1 8	If the cell is empty, then add ix to
	7) The main game function, display the
	where board get human players move,
1	for wen or draw also and hotes
-	neck last and and
+	of the moves, define a function as a
+	The state of Al words
1	Finding a beller move by keeping 50' in
	every emply cells.



2. Solve 8-Puzzle Problem (DFS)

```
Code:
```

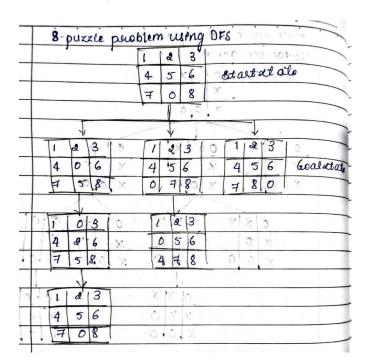
```
from collections import deque
import time
goal_state = [[1, 2, 3],
        [4, 5, 6],
        [7, 8, 0]]
def find_zero(state):
  for i in range(3):
    for j in range(3):
       if state[i][j] == 0:
         return (i, j)
def move(state, direction):
  new_state = [row[:] for row in state]
  zero_pos = find_zero(state)
  i, j = zero_pos
  if direction == "up" and i > 0:
    new_state[i][j], new_state[i-1][j] = new_state[i-1][j], new_state[i][j]
  elif direction == "down" and i < 2:
     new_state[i][j], new_state[i + 1][j] = new_state[i + 1][j], new_state[i][j]
  elif direction == "left" and j > 0:
    new_state[i][j], new_state[i][j-1] = new_state[i][j-1], new_state[i][j]
  elif direction == "right" and j < 2:
    new_state[i][j], new_state[i][j + 1] = new_state[i][j + 1], new_state[i][j]
  else:
    return None
  return new state
def is_goal(state):
  return state == goal_state
def print_state(state):
  for row in state:
    print(row)
  print("\n")
def dfs(initial_state):
  stack = [(initial_state, [])]
  visited = set()
  while stack:
    state, path = stack.pop()
    print("Exploring state in DFS:")
    print_state(state)
    if is_goal(state):
```

```
return path
    visited.add(str(state))
    for direction in ["up", "down", "left", "right"]:
       new state = move(state, direction)
       if new_state and str(new_state) not in visited:
         stack.append((new_state, path + [direction]))
  return None
# Example initial state (you can modify this)
initial_state = [[1, 2, 3],
          [4, 5, 6],
          [7, 0, 8]]
print("Initial State:")
print_state(initial_state)
start_time_dfs = time.time()
print("Solving using DFS:")
dfs solution = dfs(initial state)
end_time_dfs = time.time()
if dfs_solution:
  print("DFS Solution:", dfs_solution)
else:
  print("No solution found with DFS.")
print(f"Time taken by DFS: {end_time_dfs-start_time_dfs:.6f} seconds")
Output:
Initial State:
[1, 2, 3]
[4, 5, 6]
[7, 0, 8]
Solving using DFS:
Exploring state in DFS:
[1, 2, 3]
[4, 5, 6]
[7, 0, 8]
Exploring state in DFS:
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]
DFS Solution: ['right']
Time taken by DFS: 0.000517 seconds
```

salving 8 puzzleper	oblemi: & ibvagan
-	HATE TAKES - HE LINEYED - I
Class node:	Paint 6-1-1-14
Function_suit-(seta	te. parent=none, action=none,
path-cost=0):	4) Perlosonard DAS
SET relf- etal	le = state
SET xelf pas	rent = parent
SET Kelf. actu	on=action
SET self path	-cost = path-cost
Function expand()	,
CREATE childr	en
SET SCOLO, COL =	fend-blank()
CREATE posses	le-actions
IF evolute THEN	ADD eup' to possible-action
of now 22 the	ADD Down to possibleacte
	ADD " hell" to possible-action.
FOR action IN pa	
CREATE new	extate as a copy of delpasal

Date Page
Ten light mes whater swap mes - Haterows
The state Latter - I frame 7
ELSE IF action == Down' THEN SWAP
new state [event coll words new state [now +1) -
Marin in [col] to as a part to
ELSE IF action == " Left " THEN SWAP new date =
[nown col] with new-xtate from [(ol -1]
ELSE IP action == Right THEN SWAP NO.
xtate [21010][col] with new state[row][col+]
APPEND newnode (new-state, self action,
deef: path-cost+1) to Children.
RETURN children
FUNCTION find-blank():
FOR YOU FROM O TO Q:
and a la girl
TF XLY X THE TWO SCOL T == 0 THEN
D-1201 2000 (0)
TUNCTION death-first-xealer (Phase 2000)
goal-state): - Inode (mittal state)]
T lanties - L. wee
1241/E Quantitle es the
251 0000 = 15000000 5
15 node state == goal state
and pade
ADD suple of node, state to explored
215 Tuple of chied state Not an explosed
RETURN none (nogle):
FUNCTION point solution (node):
CDEALE DOWN
WHILE node is not None:

APPEND (node action, node state) to par	
contract and a method a filler security to part	2
HPPEND (near action)	-
SET node=node.parent	_
REVERSE POOLS	
FOR (action, state) 3N pain:	_
of action is not none then	
PRINT " Action: " action	
PRINT State	
MASS TO THE PRINT US TO SEE	
567 Andral atate = [[1,4,3],[0,4,6],[7,5]	[[[
SET goal-state = TE1, 2, 37, [4,5,6], (7,8,0]	3
SET salideon = depth first search (mitral-	
de goal state)	
2F solution is not none THEN PRINT "Solu	otion
found: " MOAT CHOOL HOT	
CALL pointsoledion (solution)	
efter ELSE Some give ap	
PRINT "solution not found"	



3. Implementing Iterative Deepening Search Algorithm.

Code:

```
from collections import deque
import time
goal_state = [[1, 2, 3],
        [4, 5, 6],
        [7, 8, 0]]
def find_zero(state):
  for i in range(3):
    for j in range(3):
       if state[i][j] == 0:
         return (i, j)
def move(state, direction):
  new_state = [row[:] for row in state]
  zero_pos = find_zero(state)
  i, j = zero_pos
  if direction == "up" and i > 0:
    new_state[i][j], new_state[i-1][j] = new_state[i-1][j], new_state[i][j]
  elif direction == "down" and i < 2:
    new_state[i][j], new_state[i + 1][j] = new_state[i + 1][j], new_state[i][j]
  elif direction == "left" and j > 0:
     new_state[i][j], new_state[i][j-1] = new_state[i][j-1], new_state[i][j]
  elif direction == "right" and j < 2:
    new_state[i][j], new_state[i][j + 1] = new_state[i][j + 1], new_state[i][j]
  else:
    return None
  return new_state
def is goal(state):
  return state == goal_state
def print_state(state):
  for row in state:
    print(row)
  print("\n")
def iterative_deepening_dfs(initial_state):
  def depth limited dfs(state, path, depth limit):
    print("Exploring state in DLS:")
    print_state(state)
    if is_goal(state):
       return path
    if depth limit == 0:
       return None
```

```
for direction in ["up", "down", "left", "right"]:
       new_state = move(state, direction)
       if new_state and str(new_state) not in visited:
         visited.add(str(new_state))
         result = depth limited dfs(new state, path + [direction], depth limit-1)
         if result is not None:
           return result
    return None
  depth limit = 0
  while True:
    visited = set()
    result = depth_limited_dfs(initial_state, [], depth_limit)
    if result is not None:
       return result
    depth_limit += 1
    if depth limit > 20: # Adjust the limit based on your puzzle complexity
       return None
# Example initial state (you can modify this)
initial\_state = [[1, 2, 3],
          [4, 5, 6],
          [7, 0, 8]]
print("Initial State:")
print_state(initial_state)
start_time_iddfs = time.time()
print("Solving using Iterative Deepening DFS:")
iddfs_solution = iterative_deepening_dfs(initial_state)
end_time_iddfs = time.time()
if iddfs solution:
  print("IDDFS Solution:", iddfs_solution)
else:
  print("No solution found with IDDFS.")
print(f"Time taken by IDDFS: {end time iddfs-start time iddfs:.6f} seconds")
Output:
Initial State:
[1, 2, 3]
[4, 5, 6]
[7, 0, 8]
Solving using Iterative Deepening DFS:
Exploring state in DLS:
[1, 2, 3]
[4, 5, 6]
[7, 0, 8]
```

Exploring state in DLS:

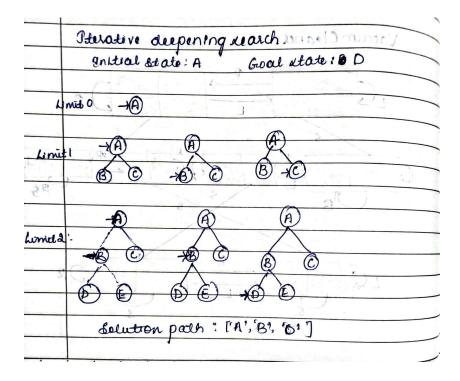
- [1, 2, 3]
- [4, 5, 6]
- [7, 0, 8]

Exploring state in DLS:

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

a)	Amplementing Iterative Deepening Bearch.
	Pseudocode: AM ADON'T = Anthrope 1 770
	FUNCTION itelative_cleepening-search (initial-state)
	goal state, depth):
	FOR depth FROM 0 to max-depth:
	SET result = depth_limited_search (initial Ha
	goal state, depth)
1	IF result to not none THEN
	CAS RETURN SHOULD MAKE ALL
suplex.	AT I'M AD RETURN none
7	FUNCTION depth-limited search (node, goal state)
	white):
	1F node state==goal-state THEN RETURN node
	IF node depth >= limit THEN RETURN none
	FOR each child an expandinodels

Pis (SET result= depth-limited_dearch (child, goal that)
	De 15 rescut is not work THEN RETURN mult
	RETURN none work orange about
SET	initial-state 83 %
SET	goal-state
	man-depth (1) (1)
SET	solution = riterative-deepening-dearch (mitial state
goal	-state, max-depth)
	solution is not none THEN PRINT solution
	E PRINT "no solution found"



4. Implement Vacuum Cleaner Agent.

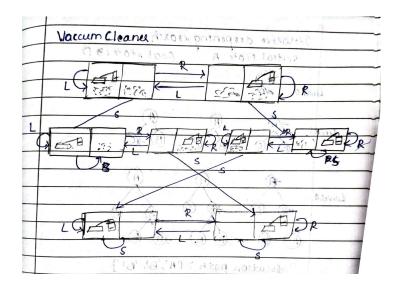
Code:

```
def vacuum_cleaner_agent(location, status):
  x, y = location
  if status[x][y] == 'Dirty':
     return f"The vacuum cleaner is at (\{x\}, \{y\}) and it is dirty. Cleaning."
     return f"The vacuum cleaner is at ({x}, {y}) and it is clean. Moving."
status = [['Dirty', 'Clean'], ['Dirty', 'Dirty']]
location = (0, 0)
while True:
  action = vacuum_cleaner_agent(location, status)
  print(action)
  x, y = location
  if status[x][y] == 'Dirty':
    status[x][y] = 'Clean'
  if status[0][0] == 'Clean' and status[0][1] == 'Clean' and status[1][0] == 'Clean' and status[1][1] == 'Clean'
'Clean':
     print("All locations are clean. The vacuum cleaner is finished.")
    break
  if y < 1:
    location = (x, y + 1)
  elif x < 1:
    location = (x + 1, 0)
```

Output:

The vacuum cleaner is at (0,0) and it is dirty. Cleaning. The vacuum cleaner is at (0,1) and it is clean. Moving. The vacuum cleaner is at (1,0) and it is dirty. Cleaning. The vacuum cleaner is at (1,1) and it is dirty. Cleaning. All locations are clean. The vacuum cleaner is finished.

	Page Payeldalor Amana Page
1	Implementing. Vaccum cleaner agent
	algorithm. The things who has a contract the
	1. well a function do state the status of
	the location, [Divity of clean]
	a. Mitialine the docation to (0,0) &
	3. Till the all the locations are cleaned,
	theck wheather the location is cleaned,
	dirty, if yes clean, else move to the next
	location.
	4. If all the locations are cleaned, state that
19.1	the docations a cleaning is finished.
	5. Elle move to the next location.



5. Implementing A* Algorithm(Misplaced Tiles)

```
Code:
```

```
#Heuristic approach to 8-puzzle problem
import heapq
def solve_8puzzle(initial_state):
  goal state = [[1, 2, 3], [8, 0, 4], [7, 6, 5]]
  priority_queue = [(heuristic(initial_state, goal_state), 0, initial_state, [])]
  visited = set()
  while priority queue:
    f_cost, g_cost, current_state, current_path = heapq.heappop(priority_queue)
    if current_state == goal_state:
       return current_path + [current_state]
    if tuple(map(tuple, current_state)) in visited:
       continue
    visited.add(tuple(map(tuple, current state)))
    for next_state, action in get_possible_moves(current_state):
       new g \cos t = g \cos t + 1
       new_f_cost = new_g_cost + heuristic(next_state, goal state)
       heapq.heappush(priority_queue, (new_f_cost, new_g_cost, next_state, current_path +
[(current_state, action)]))
  return None
def heuristic(state, goal_state):
  misplaced tiles = 0
  for i in range(3):
    for j in range(3):
       if state[i][j] != goal_state[i][j] and state[i][j] != 0:
         misplaced tiles += 1
  return misplaced_tiles
def find_position(state, tile):
  for i in range(3):
    for j in range(3):
       if state[i][j] == tile:
         return i, j
def get possible moves(state):
  row, col = find_position(state, 0)
  possible_moves = []
  if row > 0:
    new_state = [list(row) for row in state]
    new_state[row][col], new_state[row-1][col] = new_state[row-1][col], new_state[row][col]
    possible moves.append((new state, 'Up'))
  if row < 2:
    new state = [list(row) for row in state]
```

```
new_state[row][col], new_state[row + 1][col] = new_state[row + 1][col], new_state[row][col]
    possible_moves.append((new_state, 'Down'))
  if col > 0:
    new_state = [list(row) for row in state]
    new state[row][col], new state[row][col-1] = new state[row][col-1], new state[row][col]
    possible_moves.append((new_state, 'Left'))
  if col < 2:
    new_state = [list(row) for row in state]
    new_state[row][col], new_state[row][col + 1] = new_state[row][col + 1], new_state[row][col]
    possible_moves.append((new_state, 'Right'))
  return possible_moves
initial_state = [[2, 8, 3], [1, 6, 4], [0, 7, 5]]
solution = solve_8puzzle(initial_state)
if solution:
  print("Solution found:")
  for state, action in solution[:-1]:
    print("----")
    for row in state:
      print(row)
    print("Move:", action)
  print("----")
  for row in solution[-1]:
    print(row)
  print("No solution found.")
Output:
Solution found:
[2, 8, 3]
[1, 6, 4]
[0, 7, 5]
Move: Right
[2, 8, 3]
[1, 6, 4]
[7, 0, 5]
Move: Up
[2, 8, 3]
[1, 0, 4]
[7, 6, 5]
Move: Up
[2, 0, 3]
[1, 8, 4]
[7, 6, 5]
Move: Left
_____
```

[0, 2, 3]

[1, 8, 4]

[7, 6, 5]

Move: Down

[1, 2, 3]

[0, 8, 4]

[7, 6, 5]

Move: Right

[1, 2, 3]

[8, 0, 4]

[7, 6, 5]

steuristic of (mesplaced tise Pseudocode fol function solve-spurxle (millal-state, goal agaz) purorly-queue : 1 (heuristic (mit at state, goal state), O, mileal state, [])] vialled: emply set while prevolity-queue 15 net emply: · (f. wst , g-wst , weren - etate, cureen par = sumove element with lowered of court from perorely-queue of wirene-state 18 equal to goal state. selves where pack + ruisiere star of current - state B in visited: continue and current-acousto visites, for next-atate, action in get-possible. moves (current-setate): new-g-cost = g-cost +1 new-f-cost = new-g-cost + hereus to c (next-state, goal-state) add thew-f-cost, new-g-cost, nent-state werent path + 11 wevent state, action to periority queue section None. function hundlio (de ate, goal at ate). miplaced-rules = 0 for each tile in state; of tile is not in our correct position on goal state and lite is not o. muplaced . Itel : muplaced sheat relain miplaced till function get-passible-mover (detate); find position of the blank title () paulible_mover : empty eist of blank title can move up: xwap blank till with till above add (new xtate, "Up") to possible-move of blank tile can move down. awap blank tile near tile below add (new-state, "Down") to possible move of blank tile can move left: swaps blank with much tile to the left add (new state, " heft") to possible move of blank title can move night: awap blank the with the to the sight and (new at ate, "Right ") to possible mover. eletura possible_movel

	amplementing At reach algorithm Number of mesplaced little
7.	- Number of mesplaced little
	state space the
	a 8 3 g(0)=0 h(0)=5
	45
9(1)=1	
	8 8 3
	1 7 5
91212	h(a)=3 h(a)=5 h(a)=6
	8 8 3 8 3 8 8 3 8 5 3
	765 75 75
(3)≥3	b(3)=3
4	h(3)=3 h(3)=4 h(3)=1
7	8 4 14 164
141=4	65 765 765
4)=0	h(4)=3 h(4): A h(A)=3 h(4)=3 h(4)=4
· 2 3	100000000000000000000000000000000000000
8 4	1 4 1 8 4 2 1 4 1 4 7 1 4
6 5	765 765 765 765
15)=1/	
	hL5):3
	3 4 3
	184
	765
h16)=0	
i a 3	23 183
8 4	184 784
7 6 5	705 85
V	

6. Implementing A* Algorithm(Manhattan Approach)

```
Code:
```

```
#Manhattan approach
import heapq
def solve 8puzzle(initial state):
  goal_state = [[1, 2, 3], [8, 0, 4], [7, 6, 5]]
  priority queue = [(heuristic(initial state, goal state), 0, initial state, [])]
  visited = set()
  while priority_queue:
    f_cost, g_cost, current_state, current_path = heapq.heappop(priority_queue)
    if current state == goal state:
       return current_path + [current_state]
    if tuple(map(tuple, current state)) in visited:
       continue
    visited.add(tuple(map(tuple, current_state)))
    for next_state, action in get_possible_moves(current_state):
       new g \cos t = g \cos t + 1
       new_f_cost = new_g_cost + heuristic(next_state, goal_state)
       heapq.heappush(priority_queue, (new_f_cost, new_g_cost, next_state, current_path + [(current
_state, action)]))
  return None
def heuristic(state, goal state):
  distance = 0
  for i in range(3):
    for j in range(3):
       if state[i][j] != 0:
         goal row, goal col = find position(goal state, state[i][j])
         distance += abs(i - goal_row) + abs(j - goal_col)
  return distance
def find position(state, tile):
  for i in range(3):
    for j in range(3):
       if state[i][j] == tile:
         return i, j
def get_possible_moves(state):
  row, col = find position(state, 0)
  possible_moves = []
  if row > 0:
    new state = [list(row) for row in state]
    new state[row][col], new state[row - 1][col] = new state[row - 1][col], new state[row][col]
    possible_moves.append((new_state, 'Up'))
  if row < 2:
    new state = [list(row) for row in state]
    new_state[row][col], new_state[row + 1][col] = new_state[row + 1][col], new_state[row][col]
    possible_moves.append((new_state, 'Down'))
  if col > 0:
    new state = [list(row) for row in state]
    new_state[row][col], new_state[row][col - 1] = new_state[row][col - 1], new_state[row][col]
    possible_moves.append((new_state, 'Left'))
  if col < 2:
    new state = [list(row) for row in state]
    new_state[row][col], new_state[row][col + 1] = new_state[row][col + 1], new_state[row][col]
```

```
possible_moves.append((new_state, 'Right'))
  return possible_moves
initial_state = [[2, 8, 3], [1, 6, 4], [0, 7, 5]]
solution = solve_8puzzle(initial_state)
if solution:
  print("Solution found:")
  for state, action in solution[:-1]:
    print("----")
    for row in state:
      print(row)
    print("Move:", action)
  print("-----")
  for row in solution[-1]:
    print(row)
else:
  print("No solution found.")
Output:
Solution found:
-----
[2, 8, 3]
[1, 6, 4]
[0, 7, 5]
Move: Right
[2, 8, 3]
[1, 6, 4]
[7, 0, 5]
Move: Up
-----
[2, 8, 3]
[1, 0, 4]
[7, 6, 5]
Move: Up
-----
[2, 0, 3]
[1, 8, 4]
[7, 6, 5]
Move: Left
[0, 2, 3]
[1, 8, 4]
[7, 6, 5]
Move: Down
-----
[1, 2, 3]
[0, 8, 4]
[7, 6, 5]
Move: Right
[1, 2, 3]
[8, 0, 4]
[7, 6, 5]
```

function solve spexale (initial state, goal state); posibility queue = ([(manhattan_distance (snit - ral_state, goal_state); 0, snutral_state, [])] visited = empty set while perochy-queue is not empty. (f-cast, ey-cost, current state, current path) = pop smallest f-cost from queue	Preudocode for Manhatien DI	stone
pocietity queue = (I (manhattan_distance (mit -ral_state, goal_state); o, rmbral_state, [])] vrdited = empty set white perorby-queue rs not empty. (f-cast, ey-cost, current state, current path) = pop smallest f-cost from queue	function sofge-spexale (inc	lial state, goal
-ral state, goal state); 0, milral state, []) vrsiled = empty set while perorly-queue & not empty. (f-cast, g-cost, current state, current path) - pop smallest f-cost from queue	state);	
-ral state, goal state); 0, mbral state, []) vrsiled = empty set while perorely-queue re not empty. (f-cast, ej-cost, current state, current path) = pop smallest f-cost from queue !!	pourolity queue = (I manha	tlan_distance(Init
while perolly-queue is not empty. (f-cast, ey-cost, current state, current-path) - pop smallest f-cost from queue	- Palatate, goal atate); 0, mit	ral-state, [])]
- pop smallest frost from queue		
= pop smallest f.cost from queue 1)	while perolety-queue is not	imply.
= pop smallest f-cost from queue	(I wast , of well, werented	tate, cullent-path)
il issent-state== goalstate; return curint	- non smallest f-cost &	rom queue
noth + reverent-state]	il ussent_state== goal sto	ite: return current
partite	poth + [urrent_state]	

	Page	_
	if current state in visited: continue	-
	weited and current states	
rodo	for new state, action in get mover (were	eno
ණයෙ ර	to Later and Lance of the	-
	new grost = grost + 10	
1d 3	new-fcost = new g-cost + mahattan-de	tea
100	(next state, goal state)	
	add (new-f-cost, new g-cost, next-se	tate
	current part + [current atale, act	on)
to the	it 1100 queve sand apare	
	return None # No solution "	1
	chon manhattan distance extale, goal state) ;
	distance = 6 and another goods	j
- 10	for each tile in state:	
	1/ sie)=0: 10 vois	
	distance += 1 gal row - current row +	
	190alcol-currentcoll	
	return distance out A mondous	
fund	tion get move (state):	
, }	and blank tile paulian.	
	generate possaible moves (Up, Down, helps, Rig	Re)
	y swapping blank tile with adjacent tel	
	elain list of (new state, action) pairs	

			ant		F	4	8	3	1	_	hio						0 +	-1+	1+
_	Da.	3 8	3 14 5		+	10	6	4		-		-		north			+		
73. 48		11 11			+	13	1	5	7	51			_	11/2		1	_	_	_
		(i) =		3 1	1	Va		111	1	=	1) = =		_	+	-	_
_			_	=	\leq	7	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	111	113	- 14 15	10	-	8		+		+		
		8	8	1	3	-		_	_	_	-	+	6	3	+		+	_	
	Y	3.	6		4	L		3119	_		-		0 1	5	-	-	1	_	_
0	i"	· M	7	┺	5.			1 100		4	14	1	1	10	_	-	10) 2	6
4	å	head	= 4	1	11	1	-	144	-	3)=	6	13.	-				>	_	
-			d	J.	8	3	12	2,357	rti.	d	8	3	_		d	-	+	3	
'n	8	et t	nt	0) (4	a Er	of to	. 3	13	6	4	1			6	4		
1)/		334	7	3	6	5	1 6)	11	1:	117	7	5	1	Hi.	7	5			
h	3) :	3	-	11	3)2	5			10	hl	9)=	5	?	_	113)=:	5		
d	1	3	7	7	2	8	3	T	8	1 8	3		d	8	8	3		17.7	
1	8	4	9.21	1		1	4	3,5	1	4	1	1	9	6	4	4			
	_			+	-	-	-	-				_		$\overline{}$	_				
_	6		(4):	4		03	n	(4)	17	(3)	0 -1			-	6				
_		1	30	4		3 4	1	(4) a	= 4	4) } () - () () () () () () () () () () () () () () (ra.	100 E	f		2			
)=0	2 8	h 3	3 2	4	8	3 4	1	8	±4 3	4	3	ra.	100 E	1		2			
)=6	8	h 3 4 5 5	3 2	4 2	8	3 4 5	700	1 7	= 4 3 8 6	4 5) } () - () () () () () () () () () () () () () () ((d.	100 E	<i>t</i>					
)=8	2 8	h 3 4 5 5		4 2	8 1 6	3 4 5	t A	1 = 3	= 4	4 5	3 - 3 - 3 - 3 - 3 - 3 - 3 - 3 - 3 - 3 -	(), ()	aria de la constante de la con	£					
)=8	8 8	h 3 4 5 5		4 2 4	8 1 6	3 4 5	t a	1 7 3	24	4 5) 3) -7) -7) 3) 3) (b) ((), ()	aria de la constante de la con						
)=d	8 8	h 3 4 5 5		4 2 4	8 1 6	3 4 5	t4 8	3 4	± 4	4 5	10 - 10 - 10 - 10 - 10 - 10 - 10 - 10 -	10.0	132	1411	2 2 3				
)=d	3=1	h 3 4 5 5		4 2 0 7	8 1 6	3 4 5	t a	3 4	± 4	4 5	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	10.0	132	1411	2 2 3				
TA I	3 8 8 8 7 (h 3 4 5 5 4 4		4 2 0 7	8 1 6	3 4 5	t4 8	1 1 3 4 5	= 4 3 8 6	4 5	10 - 10 - 10 - 10 - 10 - 10 - 10 - 10 -	1110	132	Line of the second	2 2 3				
1 7	3 8 8 8 7 (h 3 4 5 5 4 4		4 2 0 7	8 1 6	3 4 5 1 7	8 6	3 4 5	= 4	4 5	3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2	100	100	Line of the second	2 2 3				
)=d	3=1	h 3 4 5 5 5 5 5 5		4 2 0 7	8 1 6	3 4 5 1 7	A 8 6	3 4 5	= 4	4 5	32	100	100	Line of the second	2 2 3				
)=d	3=1	h 3 4 5 5 5 5 5 5 5		4 2 0 1	8 1 6	3 4 5	A 8 6	3 4 5	= 4	4 5	3 de 10 de 1	100	1112	12.1	2 2 3				
)=d	3=1	h 3 4 4 5 5 5 5 5 4 4		4 2 0 7	8 1 6	3 4 5 1 7	14 8 6 6 3	3 4 5	= 4	4 5	3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 &	100	1112	12.1	2 2 3				