

|  |  |  |  |
| --- | --- | --- | --- |
| **LABORATORY COMPONENTS** | | | |
| **Exp. No.** | **Experiment Description** | **CO No.** | **Bloom’s Taxonomy Level** |
| 1 | Design and implement Tic-Tac-Toe game using Python programming. | CO1 | CL3 |
| 2 | Demonstrate Nim game using Python programming. | CO2 | CL3 |
| 3 | Write a Pyhton Pprogram to implement A\* Algorithm. | CO2 | CL3 |
| 4 | Write a python program to demonstrate the working of Alpha-Beta Pruning. | CO2 | CL3 |
| 5 | Demonstrate the Union and Intersection of two fuzzy Sets using python programming. | CO3 | CL3 |
| 6 | Write a program in Prolog to implement simple arithmetic. | CO4 | CL3 |
| 7 | Design and implement a Cross word puzzle using Python programming. | CO4 | CL3 |
| 8 | Demonstrate a simple Chatbot with minimum of 10 conversations. | CO5 | CL3 |

**PROGRAMS**

**1. Design and implement Tic-Tac-Toe game using Python programming**

import random

def print\_board(board):

print(" 1 2 3")

for i in range(3):

print(f"{i+1} {' '.join(board[i])}")

def check\_winner(board, player):

# Check for row wins

for row in board:

if all(cell == player for cell in row):

return True

# Check for column wins

for col in range(3):

if all(board[row][col] == player for row in range(3)):

return True

# Check for diagonal wins

if all(board[i][i] == player for i in range(3)) or all(board[i][2 - i] == player for i in range(3)):

return True

return False

def is\_board\_full(board):

return all(cell != ' ' for row in board for cell in row)

def get\_available\_moves(board):

return [(i, j) for i in range(3) for j in range(3) if board[i][j] == ' ']

def player\_move(board):

while True:

try:

row, col = map(int, input("Enter your move (row and column, e.g., 1 2): ").split())

if 1 <= row <= 3 and 1 <= col <= 3 and board[row - 1][col - 1] == ' ':

return row - 1, col - 1

else:

print("Invalid move. Try again.")

except ValueError:

print("Invalid input. Please enter two integers separated by a space.")

def computer\_move(board, computer, player):

available\_moves = get\_available\_moves(board)

# Try to win

for move in available\_moves:

board[move[0]][move[1]] = computer

if check\_winner(board, computer):

return move

board[move[0]][move[1]] = ' ' # Undo move

# Block player from winning

for move in available\_moves:

board[move[0]][move[1]] = player

if check\_winner(board, player):

board[move[0]][move[1]] = computer

return move

board[move[0]][move[1]] = ' ' # Undo move

# Otherwise, make a random move

return random.choice(available\_moves)

def play\_game():

#Creating a 2D array board with string value ' ' in it.

board = [[' ' for i in range(3)] for j in range(3)]

#board = [[ , , ]

# [ , , ]

# [ , , ]]

player = 'X'

computer = 'O'

while True:

print\_board(board) #initial board

if check\_winner(board, player):

print("Congratulations! You win!")

break

elif check\_winner(board, computer):

print("Computer wins!")

break

elif is\_board\_full(board):

print("It's a tie!")

break

if player == 'X':

row, col = player\_move(board)

board[row][col] = 'X'

player = 'O'

computer = 'X'

else:

row, col = computer\_move(board, computer, player)

board[row][col] = 'O'

player = 'X'

computer = 'O'

if \_\_name\_\_ == "\_\_main\_\_":

play\_game()

**2.Demonstrate Nim game using Python programming.**

def print\_board(heap):

print(f"Current heap: {'|' \* heap}")

def get\_user\_move(heap):

while True:

try:

sticks\_to\_remove = int(input(f"Enter the number of sticks to remove (minimum 1, maximum {min(heap, heap // 2)}): "))

if 1 <= sticks\_to\_remove <= min(heap, heap // 2):

break

else:

print(f"Invalid number of sticks. Please enter a number between 1 and {min(heap, heap // 2)}.")

except ValueError:

print("Invalid input. Please enter a valid number.")

return sticks\_to\_remove

def get\_computer\_move(heap):

xor\_sum = heap

for i in range(heap):

xor\_sum ^= i

if xor\_sum == 0:

max\_sticks = min(heap, heap // 2)

sticks\_to\_remove = random.randint(1, max\_sticks)

else:

max\_sticks = min(heap // 2, heap)

sticks\_to\_remove = max(1, min(max\_sticks, heap - xor\_sum))

return sticks\_to\_remove

def nim\_game():

heap = 16

player\_turn = 1

while heap > 1:

print\_board(heap)

if player\_turn == 1:

player\_name = "Player 1"

sticks\_to\_remove = get\_user\_move(heap)

else:

player\_name = "Computer"

sticks\_to\_remove = get\_computer\_move(heap)

heap -= sticks\_to\_remove

print(f"{player\_name} removes {sticks\_to\_remove} sticks.")

player\_turn = 3 - player\_turn # Switch player (1 -> 2, 2 -> 1)

print\_board(heap)

winner = "Player 1" if player\_turn == 2 else "Computer"

print(f"Player {player\_turn} picks the last stick ")

print(f"\n{winner} is the winner!")

if \_\_name\_\_ == "\_\_main\_\_":

import random

nim\_game()

**3.Write a Pyhton Pprogram to implement A\* Algorithm**

def aStarAlgo(start\_node, stop\_node):

open\_set = set(start\_node)

closed\_set = set()

g = {} #store distance from starting node

parents = {}# parents contains an adjacency map of all nodes

#ditance of starting node from itself is zero

g[start\_node] = 0

#start\_node is root node i.e it has no parent nodes

#so start\_node is set to its own parent node

parents[start\_node] = start\_node

while len(open\_set) > 0:

n = None

#node with lowest f() is found

for v in open\_set:

if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):

n = v

if n == stop\_node or Graph\_nodes[n] == None:

pass

else:

for (m, weight) in get\_neighbors(n):

#nodes 'm' not in first and last set are added to first

#n is set its parent

if m not in open\_set and m not in closed\_set:

open\_set.add(m)

parents[m] = n

g[m] = g[n] + weight

#for each node m,compare its distance from start i.e g(m) to the

#from start through n node

else:

if g[m] > g[n] + weight:

#update g(m)

g[m] = g[n] + weight

#change parent of m to n

parents[m] = n

#if m in closed set,remove and add to open

if m in closed\_set:

closed\_set.remove(m)

open\_set.add(m)

if n == None:

print('Path does not exist!')

return None

# if the current node is the stop\_node

# then we begin reconstructin the path from it to the start\_node

if n == stop\_node:

path = []

while parents[n] != n:

path.append(n)

n = parents[n]

path.append(start\_node)

path.reverse()

print('Path found: {}'.format(path))

return path

# remove n from the open\_list, and add it to closed\_list

# because all of his neighbors were inspected

open\_set.remove(n)

closed\_set.add(n)

print('Path does not exist!')

return None

#define fuction to return neighbor and its distance

#from the passed node

def get\_neighbors(v):

if v in Graph\_nodes:

return Graph\_nodes[v]

else:

return None

#for simplicity we ll consider heuristic distances given

#and this function returns heuristic distance for all nodes

def heuristic(n):

H\_dist = {

'A': 11,

'B': 6,

'C': 99,

'D': 1,

'E': 7,

'G': 0,

}

return H\_dist[n]

#Describe your graph here

Graph\_nodes = {

'A': [('B', 2), ('E', 3)],

'B': [('C', 1),('G', 9)],

'C': None,

'E': [('D', 6)],

'D': [('G', 1)],

}

aStarAlgo('A', 'G')

4.Write a python program to demonstrate the working of Alpha-Beta Pruning

MAX, MIN = 1000, -1000

def minimax(depth, nodeIndex, maximizingPlayer,

values, alpha, beta):

if depth == 3:

return values[nodeIndex]

if maximizingPlayer:

best = MIN

for i in range(0, 2):

val = minimax(depth + 1, nodeIndex \* 2 + i, False, values, alpha, beta)

best = max(best, val)

alpha = max(alpha, best)

# Alpha Beta Pruning

if beta <= alpha:

break

return best

else:

best = MAX

for i in range(0, 2):

val = minimax(depth + 1, nodeIndex \* 2 + i, True, values, alpha, beta)

best = min(best, val)

beta = min(beta, best)

# Alpha Beta Pruning

if beta <= alpha:

break

return best

# Driver Code

if \_\_name\_\_ == "\_\_main\_\_":

values = [10,9,14,18,5,4,50,3]

print("The optimal value is :", minimax(0, 0, True, values, MIN, MAX))

5. Demonstrate the Union and Intersection of two fuzzy Sets using python programming.

def fuzzy\_union(set1, set2):

union\_set = {}

for element in set1:

union\_set[element] = max(set1[element], set2.get(element, 0))

for element in set2:

if element not in union\_set:

union\_set[element] = set2[element]

return union\_set

def fuzzy\_intersection(set1, set2):

intersection\_set = {}

for element in set1:

if element in set2:

intersection\_set[element] = min(set1[element], set2[element])

return intersection\_set

def display\_fuzzy\_set(fuzzy\_set):

print("{", end="")

for element, membership in fuzzy\_set.items():

print(f"{element}: {membership}", end=", ")

print("}")

# Example fuzzy sets

set1 = {'a': 0.8, 'b': 0.6, 'c': 0.4, 'd': 0.2, 'e': 0.1}

set2 = {'a': 0.7, 'b': 0.5, 'c': 0.3, 'f': 0.9, 'g': 0.4}

print("Fuzzy set 1:")

display\_fuzzy\_set(set1)

print("\nFuzzy set 2:")

display\_fuzzy\_set(set2)

print("\nUnion of the fuzzy sets:")

union\_set = fuzzy\_union(set1, set2)

display\_fuzzy\_set(union\_set)

print("\nIntersection of the fuzzy sets:")

intersection\_set = fuzzy\_intersection(set1, set2)

display\_fuzzy\_set(intersection\_set)

**6. Write a program in Prolog to implement simple arithmetic.**

**7. Design and implement a Cross word puzzle using Python programming.**

import random

class Crossword:

def \_\_init\_\_(self, height, width):

self.height = height

self.width = width

self.grid = [[' ' for \_ in range(width)] for \_ in range(height)]

self.words = []

def add\_word(self, word):

word = word.upper()

if len(word) > max(self.height, self.width):

print(f"Word '{word}' is too long to fit in the grid.")

return

self.words.append(word)

placed = False

while not placed:

direction = random.choice(['across', 'down'])

if direction == 'across':

x = random.randint(0, self.width - len(word))

y = random.randint(0, self.height - 1)

if self.check\_fit(word, x, y, 1, 0):

self.place\_word(word, x, y, 1, 0)

placed = True

else:

x = random.randint(0, self.width - 1)

y = random.randint(0, self.height - len(word))

if self.check\_fit(word, x, y, 0, 1):

self.place\_word(word, x, y, 0, 1)

placed = True

def check\_fit(self, word, x, y, dx, dy):

for i in range(len(word)):

if self.grid[y][x] not in [' ', word[i]]:

return False

x += dx

y += dy

return True

def place\_word(self, word, x, y, dx, dy):

for i in range(len(word)):

self.grid[y][x] = word[i]

x += dx

y += dy

def display(self):

for row in self.grid:

print(' '.join(row))

def main():

crossword = Crossword(12, 12)

words = ["PYTHON", "ALGORITHM", "PROGRAMMING", "COMPUTER", "LANGUAGE"]

for word in words:

crossword.add\_word(word)

crossword.display()

if \_\_name\_\_ == "\_\_main\_\_":

main()

**8. Demonstrate a simple Chatbot with minimum of 10 conversations.**

import random

# Dictionary containing responses for different user inputs

responses = {

"hi": ["Hello!", "Hi there!", "Hey!"],

"how are you": ["I'm doing well, thank you!", "I'm great, thanks for asking!"],

"what's your name": ["I'm a chatbot!", "You can call me ChatBot."],

"what do you do": ["I'm here to chat with you!", "I'm a conversational agent."],

"bye": ["Goodbye!", "See you later!", "Bye! Have a great day!"],

"tell me a joke": ["Why don't scientists trust atoms? Because they make up everything!", "I'm not good at jokes, but here's one: Why was the math book sad? Because it had too many problems."],

"who created you": ["I was created by a team of developers.", "My creators prefer to remain anonymous."],

"what is the weather today": ["I'm sorry, I can't provide real-time information.", "You can check the weather on a weather website or app."],

"how old are you": ["I don't have an age. I'm just a program!", "I exist in the realm of ones and zeros, so I don't age."],

"what is the meaning of life": ["The meaning of life is a philosophical question that has puzzled humans for centuries.", "I think the meaning of life is subjective and varies from person to person."]

}

# Function to get response from the bot

def get\_response(user\_input):

if user\_input.lower() in responses:

return random.choice(responses[user\_input.lower()])

else:

return "I'm sorry, I don't understand that."

# Main function to run the chatbot

def main():

print("Welcome to the ChatBot!")

print("You can start chatting. Type 'bye' to exit.")

while True:

user\_input = input("You: ")

if user\_input.lower() == 'bye':

print(get\_response(user\_input))

break

else:

print("ChatBot:", get\_response(user\_input))

if \_\_name\_\_ == "\_\_main\_\_":

main()