Experiment - 4

Name: - Devansh Shukla
Prn No: - 20200802109
Track: - AI

Aim:

Program to solve Missionaries-Cannibals Problem.

Software Required:

PyCharm

Theory:

In the missionaries and cannibals' problem, three missionaries and three cannibals must cross a river using a boat which can carry at most two people, under the constraint that, for both banks, if there are missionaries present on the bank, they cannot be outnumbered by cannibals (if they were, the cannibals would eat the missionaries). The boat cannot cross the river by itself with no people on board.

Algorithm:

```python
from queue import Queue


# Define the state class to represent the state of the problem
class State:
    def __init__(self, missionaries, cannibals, boat_position):
        self.missionaries = missionaries
        self.cannibals = cannibals
        self.boat_position = boat_position


# Check if a state is valid
def is_valid(state):
    if state.missionaries < 0 or state.missionaries > 3 or state.cannibals < 0 or state.cannibals > 3:
        return False
    if state.missionaries < state.cannibals and state.missionaries > 0:
        return False
    if 3 - state.missionaries < 3 - state.cannibals and 3 - state.missionaries > 0:
        return False
    return True


# Check if a state is the goal state
def is_goal(state):
    return state.missionaries == 0 and state.cannibals == 0 and state.boat_position == 0
```

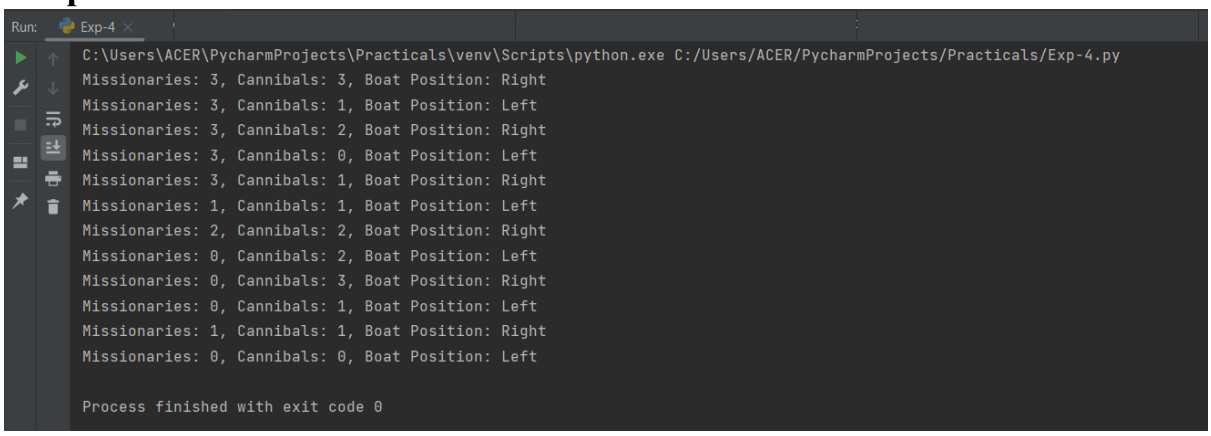
```python
# Breadth-First Search to solve the problem
def bfs():
    start_state = State(3, 3, 1)
    if is_goal(start_state):
        return [start_state]
    visited = set()
    q = Queue()
    q.put([start_state])
    while not q.empty():
        path = q.get()
        state = path[-1]
        for next_state in get_next_states(state):
            if is_valid(next_state) and next_state not in visited:
                new_path = list(path)
                new_path.append(next_state)
                if is_goal(next_state):
                    return new_path
                q.put(new_path)
                visited.add(next_state)
    return None
```

```python
# Get possible next states from the current state
def get_next_states(state):
    next_states = []
    moves = [(1, 0), (2, 0), (0, 1), (0, 2), (1, 1)]
    for move in moves:
        if state.boat_position == 1:
            new_state = State(state.missionaries - move[0], state.cannibals - move[1], 0)
        else:
            new_state = State(state.missionaries + move[0], state.cannibals + move[1], 1)
        new_state.parent = state
        next_states.append(new_state)
    return next_states


# Print the solution path
def print_solution(path):
    for t in path:
        print(
            f'Missionaries: {t.missionaries}, Cannibals: {t.cannibals}, Boat Position: {"Left" if t.boat_position == 0 else "Right"}')


# Solve the problem and print the solution
path = bfs()
if path:
    print_solution(path)
else:
    print("No solution found.")
```

Applications:

The Missionaries and Cannibals problem is a classic puzzle in Artificial Intelligence and Computer Science, often used to illustrate the concepts of problem-solving, search algorithms, and state space exploration.

Output:


Run:
Exp-4
C:\Users\ACER\PycharmProjects\Practicals\venv\Scripts\python.exe C:/Users/ACER/PycharmProjects/Practicals/Exp-4.py
Missionaries: 3, Cannibals: 3, Boat Position: Right
Missionaries: 3, Cannibals: 1, Boat Position: Left
Missionaries: 3, Cannibals: 2, Boat Position: Right
Missionaries: 3, Cannibals: 0, Boat Position: Left
Missionaries: 3, Cannibals: 1, Boat Position: Right
Missionaries: 1, Cannibals: 1, Boat Position: Left
Missionaries: 2, Cannibals: 2, Boat Position: Right
Missionaries: 0, Cannibals: 2, Boat Position: Left
Missionaries: 0, Cannibals: 3, Boat Position: Right
Missionaries: 0, Cannibals: 1, Boat Position: Left
Missionaries: 1, Cannibals: 1, Boat Position: Right
Missionaries: 0, Cannibals: 0, Boat Position: Left
Process finished with exit code 0