VIT Vidyalankar Institute of Technology Accredited A+ by NAAC	**DEPARTMENT OF COMPUTER ENGINEERING**

Experiment No. 5

Semester	T.E. Semester VI
Subject	ARTIFICIAL INTELLIGENCE (CSL 604)
Subject Professor In-charge	Prof. Avinash Shrivas
Assisting Teachers	Prof. Avinash Shrivas

Student Name	Deep Salunkhe
Roll Number	21102A0014
Lab Number	310A

Title:

Finding the path in the maze using A*

Theory:

1. Initialization: Start with the initial node (start point) and initialize its cost to reach from the start node and its heuristic estimate of the cost to reach the goal node.
2. Expansion: A* maintains a priority queue of nodes to be expanded. At each step, it expands the node with the lowest total estimated cost (the sum of the cost to reach the node from the start plus the heuristic estimate of the cost to reach the goal).
3. Evaluation of Neighbors: For each neighboring node of the current node, calculate its total estimated cost and add it to the priority queue if it has not been visited or if the new cost is lower than the previously calculated cost.
4. Termination: Repeat the expansion process until the goal node is reached or the priority queue is empty.
5. Path Reconstruction: Once the goal node is reached, trace back the path from the goal node to the start node using the parent pointers stored during the expansion process.

A* is guaranteed to find the shortest path if certain conditions are met:

- The heuristic function used must be admissible, meaning it never overestimates the cost to reach the goal node.
- The graph or search space must be finite, and the cost of each edge or step must be non-negative.

Program Code:

```
#include<iostream>
#include<vector>
#include<math.h>
using namespace std;


void printmaze(vector<vector<float> >maze){
    for(int i=0;i<4;i++){
        for(int j=0;j<8;j++){
            cout<<float(maze[i][j])<<"        ";
        }
        cout<<endl;
        cout<<endl;
        cout<<endl;
    }

    return ;
}


float EU(int ri,int ci,int rf,int cf){
    float val=0;
    val=sqrt(abs(ri-rf)*abs(ri-rf)+abs(ci-cf)*abs(ci-cf));
    return val;
}

float MH(int ri,int ci,int rf,int cf){
    float val=0;
    val=abs(ri-rf)+abs(ci-cf);
    return val;
}

void calf(vector<vector<float> >&maze){
        for(int i=0;i<4;i++){
        for(int j=0;j<8;j++){
            if(maze[i][j]==-1) continue;

            maze[i][j]=max(i,j)+EU(i,j,3,7);
        }

    }
}

void findpath(vector<vector<float> >maze,vector<vector<int> >&visted){
    vector<vector<int> >dir={{1,1},{1,0},{0,1},{0,-1},{-1,0},{-1,-1},{-1,1},{1,-
1}};
```

```
    int isr=0;
    int isc=0;
    int fsr=3;
    int fsc=7;
    float minf=maze[0][0];

    cout<<"("<<isr<<","<<isc<<")->";

    while(isr!=3 || isc!=7){ // Changed '&&' to '||' in the loop condition
       for(auto x:dir){
            int nr=isr+x[0];
            int nc=isc+x[1]; // Corrected the variable used for column index
            //cout<<nr<<" "<<nc<<endl;
            if(nr<=3 && nr>=0 && nc<=7 && nc>=0 && maze[nr][nc]!=-1.0000 &&
visted[nr][nc]==0){
                //cout<<"("<<nr<<","<<nc<<")->";
                //cout<<maze[nr][nc]<<endl;
                //cout<<"Deep Salunkhe"<<endl;
                if(maze[nr][nc]<=minf){

                    //mark as visited
                    visted[isr][isc]=1;
                    minf=maze[nr][nc];
                    isr=nr;
                    isc=nc;
                    cout<<"("<<isr<<","<<isc<<")->";
                }
            }
       }
    }

    cout<<"("<<isr<<","<<isc<<")"<<endl;
}


int main(){

  vector<vector<float> >maze(4,vector<float>(8,0));
  maze[2][0]=-1.0000;
  maze[3][0]=-1.0000;
  maze[2][1]=-1.0000;
  maze[2][3]=-1.0000;
  maze[0][4]=-1.0000;
  maze[2][4]=-1.0000;
  maze[0][5]=-1.0000;
  maze[1][5]=-1.0000;
```

```
  maze[0][6]=-1.0000;
  maze[1][6]=-1.0000;
  maze[0][7]=-1.0000;
  maze[1][7]=-1.0000;

  //definging a visterd vector
    vector<vector<int> >visted(4,vector<int>(8,0));

  cout<<"The Maze is"<<endl;
  printmaze(maze);
  cout<<"lets build the f valued matrix"<<endl;
  calf(maze);

  printmaze(maze);
  findpath(maze,visted);

  return 0;


}
```

Output:

```
PS E:\GIt\SEM-6> cd "e:\GIt\SEM-6\AI\" ; if ($?) { g++ Lab5_A.cpp -o Lab5_A } ; if ($?) { .\Lab5_A }
The Maze is
0       0       0       0       -1      -1      -1      -1


0       0       0       0       0       -1      -1      -1


-1      -1      0       -1      -1      0       0       0


-1      0       0       0       0       0       0       0


lets build the f valued matrix
7.61577         7.7082          7.83095         8       -1      -1      -1      -1


8.28011         7.32456         7.38516         7.47214         7.60555         -1      -1      -1


-1      -1      7.09902         -1      -1      7.23607         7.41421         8


-1      9       8       7       7       7       7       7


(0,0)->(1,1)->(2,2)->(3,3)->(3,4)->(3,5)->(3,6)->(3,7)->(3,7)
PS E:\GIt\SEM-6\AI>
```

Conclusion:

A* is widely used in various applications such as robotics, video games, route planning, and maze solving due to its efficiency and ability to find optimal solutions in many scenarios. In your maze-solving problem, A* can efficiently find the shortest path through the maze by considering both the actual cost of moving from one cell to another and the estimated cost to reach the goal from each cell, using a suitable heuristic such as Euclidean distance or Manhattan distance.