Assignment -3

Informed Search :

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

// Define the size of the grid

#define ROWS 4

#define COLS 4

char output [ROWS] [COLS];

// Define the structure for a node in the grid

typedef struct Node {

int x, y; // Coordinates

int f, g, h; // A\* specific values

struct Node\* parent;

} Node;

// Function to check if a given cell is within the grid

bool isValid(int x, int y) {

return (x >= 0 && x < ROWS && y >= 0 && y < COLS);

}

// Function to check if a given cell is blocked (obstacle)

bool isBlocked(int grid[ROWS][COLS], int x, int y) {

return (grid[x][y] == 1);

}

// Function to calculate the heuristic value (Manhattan distance)

int calculateHValue(int x1, int y1, int x2, int y2) {

return abs(x1 - x2) + abs(y1 - y2);

}

// Function to implement the A\* algorithm

void AStar(int grid[ROWS][COLS], Node\* start, Node\* goal) {

// Define the 4 possible movement directions: up, down, left, right

int dx[] = {-1, 1, 0, 0};

int dy[] = {0, 0, -1, 1};

// Create open and closed lists

Node\* openList[ROWS \* COLS];

Node\* closedList[ROWS \* COLS];

// Initialize counts and lists

int openCount = 0, closedCount = 0;

openList[openCount++] = start;

while (openCount > 0) {

// Find the node with the lowest f value in the open list

/\*for (int j = 0; j < closedCount; j++) {

printf("%d %d %d %d %d\n",closedList[j]->x,closedList[j]->y,closedList[j]->f,closedList[j]->g,closedList[j]->h);

}

printf("\n");\*/

int currentIndex = 0;

for (int i = 0; i < openCount; i++) {

if (openList[i]->f < openList[currentIndex]->f) {

currentIndex = i;

}

}

Node\* current = openList[currentIndex];

// Move the current node from open list to closed list

openList[currentIndex] = openList[--openCount];

closedList[closedCount++] = current;

// Check if the goal has been reached

if (current->x == goal->x && current->y == goal->y) {

// Path found, reconstruct and print it

/\* for(int i=0;i<ROWS;i++){

for(int j=0;j<COLS;j++){

output[i][j]=(char)grid[i][j];

}

}

for(int i=0;i<ROWS;i++){

for(int j=0;j<COLS;j++){

printf("%c ",output[i][j]);

}

printf("\n");

}\*/

printf("Path found:\n");

while (current != NULL) {

printf("(%d, %d) ", current->x, current->y);

//output[current->x][current->y]='\*';

current = current->parent;

}

return;

}

// Explore neighbors

for (int i = 0; i < 4; i++) {

int newX = current->x + dx[i];

int newY = current->y + dy[i];

if (isValid(newX, newY) && !isBlocked(grid, newX, newY)) {

// Create a neighbor node

Node\* neighbor = (Node\*)malloc(sizeof(Node));

neighbor->x = newX;

neighbor->y = newY;

neighbor->parent = current;

neighbor->g = current->g + 1;

neighbor->h = calculateHValue(newX, newY, goal->x, goal->y);

neighbor->f = neighbor->g + neighbor->h;

//printf("%d %d %d %d %d\n",neighbor->x,neighbor->y,neighbor->f,neighbor->g, neighbor->h);

// Check if the neighbor is already in the closed list

bool inClosedList = false;

for (int j = 0; j < closedCount; j++) {

if (closedList[j]->x == neighbor->x && closedList[j]->y == neighbor->y) {

inClosedList = true;

break;

}

}

if (!inClosedList) {

// Check if the neighbor is already in the open list

bool inOpenList = false;

for (int j = 0; j < openCount; j++) {

if (openList[j]->x == neighbor->x && openList[j]->y == neighbor->y) {

inOpenList = true;

break;

}

}

if (!inOpenList) {

openList[openCount++] = neighbor;

}

}

}

}

}

// No path found

printf("No path found!\n");

}

/\*void print(char grid[ROWS][COLS]){

printf(" --- --- --- --- --- --- --- --- \n");

for(int i=0;i<ROWS;i++){

printf("|");

for(int j=0;j<COLS;j++){

printf(" %c |",grid[i][j]);

}

printf("\n --- --- --- --- --- --- --- --- \n");

}

}\*/

int main() {

/\*int grid[ROWS][COLS] = {

{0, 0, 1, 0, 0},

{0, 0, 1, 0, 0},

{0, 0, 0, 0, 0},

{0, 1, 1, 1, 0},

{0, 0, 0, 0, 0}

};\*/

int grid[ROWS][COLS] = {

{0, 0, 0, 1},

{1, 0, 1, 0},

{1, 0, 0, 1},

{1, 1, 0, 0}

};

Node start = {0, 0, 0, 0, 0, NULL};

Node goal = {3, 3, 0, 0, 0, NULL};

AStar(grid, &start, &goal);

printf("\n");

/\*for(int i=0;i<ROWS;i++){

for(int j=0;j<COLS;j++){

printf("%c ",output[i][j]);

}

printf("\n");

}\*/

return 0;

}