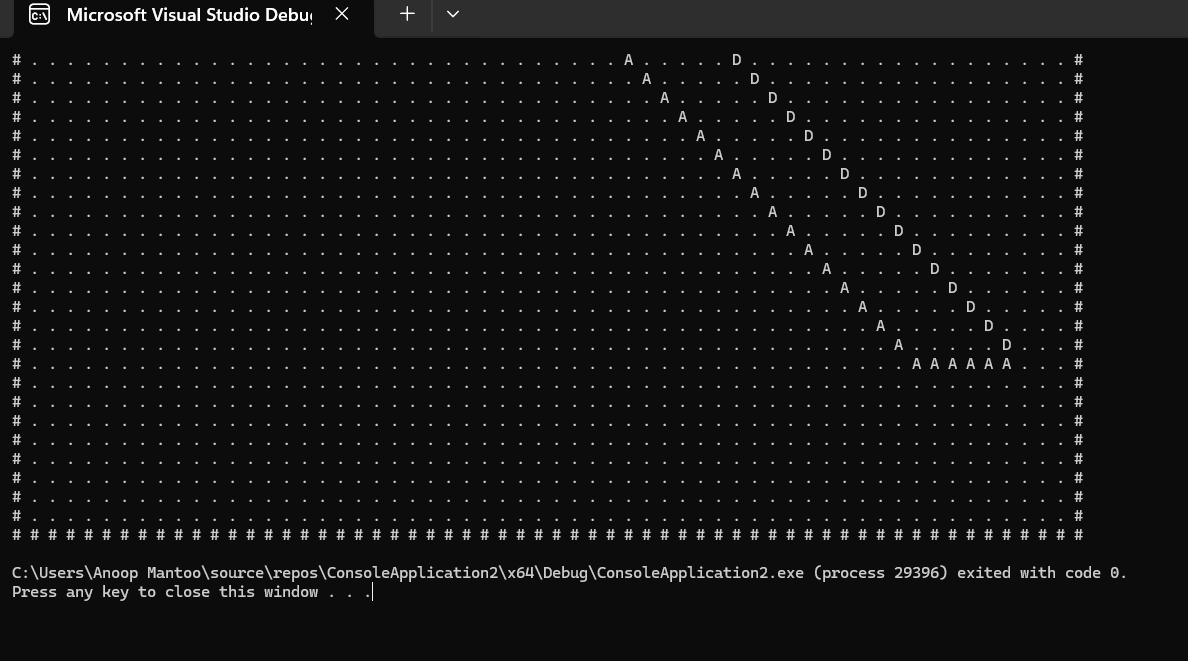
**OUTPUT:**



**SOURCE CODE [CONSOLE BASED]:**

#include <iostream>

#include <vector>

#include <queue>

#include <cmath>

#include <limits>

#include <set>

using namespace std;

#define num 60

// Dijkstra's Pathfinding

vector<pair<int, int>> pathD;

bool sptSet[num][num];

void findmin(float dist[num][num], int& min\_x, int& min\_y) {

float mini = numeric\_limits<float>::max();

for (int i = 0; i < num; i++)

for (int j = 0; j < num; j++)

if (!sptSet[i][j] && dist[i][j] < mini) {

mini = dist[i][j];

min\_x = i;

min\_y = j;

}

}

void findpath(pair<int, int> previous[num][num], float dist[num][num], int dest\_x, int dest\_y, int source\_x, int source\_y) {

cout << "\nLength of Dijkstra path is: " << dist[dest\_x][dest\_y] << endl;

while (previous[dest\_x][dest\_y].first != source\_x || previous[dest\_x][dest\_y].second != source\_y) {

pathD.push\_back(make\_pair(previous[dest\_x][dest\_y].first, previous[dest\_x][dest\_y].second));

int save\_x = dest\_x, save\_y = dest\_y;

dest\_x = previous[save\_x][save\_y].first;

dest\_y = previous[save\_x][save\_y].second;

}

}

void dijkstra(int source\_x, int source\_y, int dest\_x, int dest\_y, int grid[num][num]) {

pair<int, int> previous[num][num];

float dist[num][num];

for (int i = 0; i < num; i++)

for (int j = 0; j < num; j++)

dist[i][j] = numeric\_limits<float>::max();

dist[source\_x][source\_y] = 0.0;

int found = 0;

for (int i = 0; i < num && found == 0; i++) {

for (int j = 0; j < num && found == 0; j++) {

int min\_x = 0, min\_y = 0;

findmin(dist, min\_x, min\_y);

sptSet[min\_x][min\_y] = true;

if (sptSet[dest\_x][dest\_y] == true) {

found = 1;

break;

}

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

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

for (int i = 0; i < 8; ++i) {

int newRow = min\_x + possibleX[i];

int newCol = min\_y + possibleY[i];

if (newRow >= 0 && newRow < num && newCol >= 0 && newCol < num && grid[newRow][newCol] == 1 && !sptSet[newRow][newCol] && dist[newRow][newCol] > dist[min\_x][min\_y] + 1.0) {

dist[newRow][newCol] = dist[min\_x][min\_y] + 1.0;

previous[newRow][newCol] = make\_pair(min\_x, min\_y);

}

}

}

}

findpath(previous, dist, dest\_x, dest\_y, source\_x, source\_y);

}

// A\* Pathfinding

typedef pair<int, int> Pair;

typedef pair<float, pair<int, int>> Ppair;

bool closedList[num][num];

vector<Pair> pathA;

struct cell {

int parent\_x, parent\_y;

float f, g, h;

cell() : f(numeric\_limits<float>::max()), g(numeric\_limits<float>::max()), h(numeric\_limits<float>::max()), parent\_x(-1), parent\_y(-1) {};

};

bool isDestination(int row, int col, Pair dest) {

return row == dest.first && col == dest.second;

}

float calculateHvalue(int row, int col, Pair dest) {

int dx = abs(dest.first - row);

int dy = abs(dest.second - col);

return abs(dx - dy) + sqrt(2) \* min(dx, dy);

}

void tracePath(Pair source, Pair dest, cell cellDetails[][num]) {

int i = cellDetails[dest.first][dest.second].parent\_x, j = cellDetails[dest.first][dest.second].parent\_y;

while (!(i == source.first && j == source.second)) {

pathA.push\_back(make\_pair(i, j));

int temp\_i = i;

int temp\_j = j;

i = cellDetails[temp\_i][temp\_j].parent\_x;

j = cellDetails[temp\_i][temp\_j].parent\_y;

}

cout << "\nLength of A\* path(g) is: " << cellDetails[dest.first][dest.second].g << endl;

}

void Astar(Pair source, Pair dest, int grid[][num]) {

set<Ppair> openList;

cell cellDetails[num][num];

int i = source.first, j = source.second;

cellDetails[i][j].f = 0.0;

cellDetails[i][j].g = 0.0;

cellDetails[i][j].h = 0.0;

cellDetails[i][j].parent\_x = i;

cellDetails[i][j].parent\_y = j;

openList.insert(make\_pair(0.0, make\_pair(i, j)));

bool destFound = false;

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

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

while (!openList.empty()) {

Ppair p = \*openList.begin();

openList.erase(openList.begin());

int i = p.second.first, j = p.second.second;

closedList[i][j] = true;

if (isDestination(i, j, dest)) {

destFound = true;

break;

}

for (int k = 0; k < 8; ++k) {

int newRow = i + possibleX[k];

int newCol = j + possibleY[k];

if (newRow >= 0 && newRow < num && newCol >= 0 && newCol < num && grid[newRow][newCol] == 1 && !closedList[newRow][newCol]) {

cell successor;

successor.g = cellDetails[i][j].g + 1.0;

successor.h = calculateHvalue(newRow, newCol, dest);

successor.f = successor.g + successor.h;

successor.parent\_x = i;

successor.parent\_y = j;

if (cellDetails[newRow][newCol].g > successor.g) {

cellDetails[newRow][newCol] = successor;

openList.insert(make\_pair(successor.f, make\_pair(newRow, newCol)));

}

}

}

}

if (!destFound)

cout << "Destination cell not found.\n";

else

tracePath(source, dest, cellDetails);

}

// Main function

int main() {

int filled[num][num];

int grid[num][num];

for (int i = 0; i < num; i++)

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

if (i == 0 || i == num - 1 || j == 0 || j == num - 1)

grid[i][j] = 0;

else

grid[i][j] = 1;

}

for (int i = 0; i < num; i++)

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

sptSet[i][j] = false;

filled[i][j] = 0;

}

memset(closedList, false, sizeof(closedList));

int source\_x = 2, source\_y = 2, dest\_x = 50, dest\_y = 56;

// Run Dijkstra's and A\* algorithms

dijkstra(source\_x, source\_y, dest\_x, dest\_y, grid);

Astar(make\_pair(source\_x, source\_y), make\_pair(dest\_x, dest\_y), grid);

// Display results in console

cout << "Grid with paths:\n";

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

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

if (find(pathD.begin(), pathD.end(), make\_pair(i, j)) != pathD.end())

cout << "D "; // Path found by Dijkstra's

else if (find(pathA.begin(), pathA.end(), make\_pair(i, j)) != pathA.end())

cout << "A "; // Path found by A\*

else if (grid[i][j] == 0)

cout << "# "; // Obstacle

else

cout << ". "; // Empty space

}

cout << endl;

}

return 0;

}